アスキー・システム・バンク

PC-8801 マシン語入門

藤井敬雄・民田雅人共著

PC88♯3

ASCII SYSTEM'S BANK

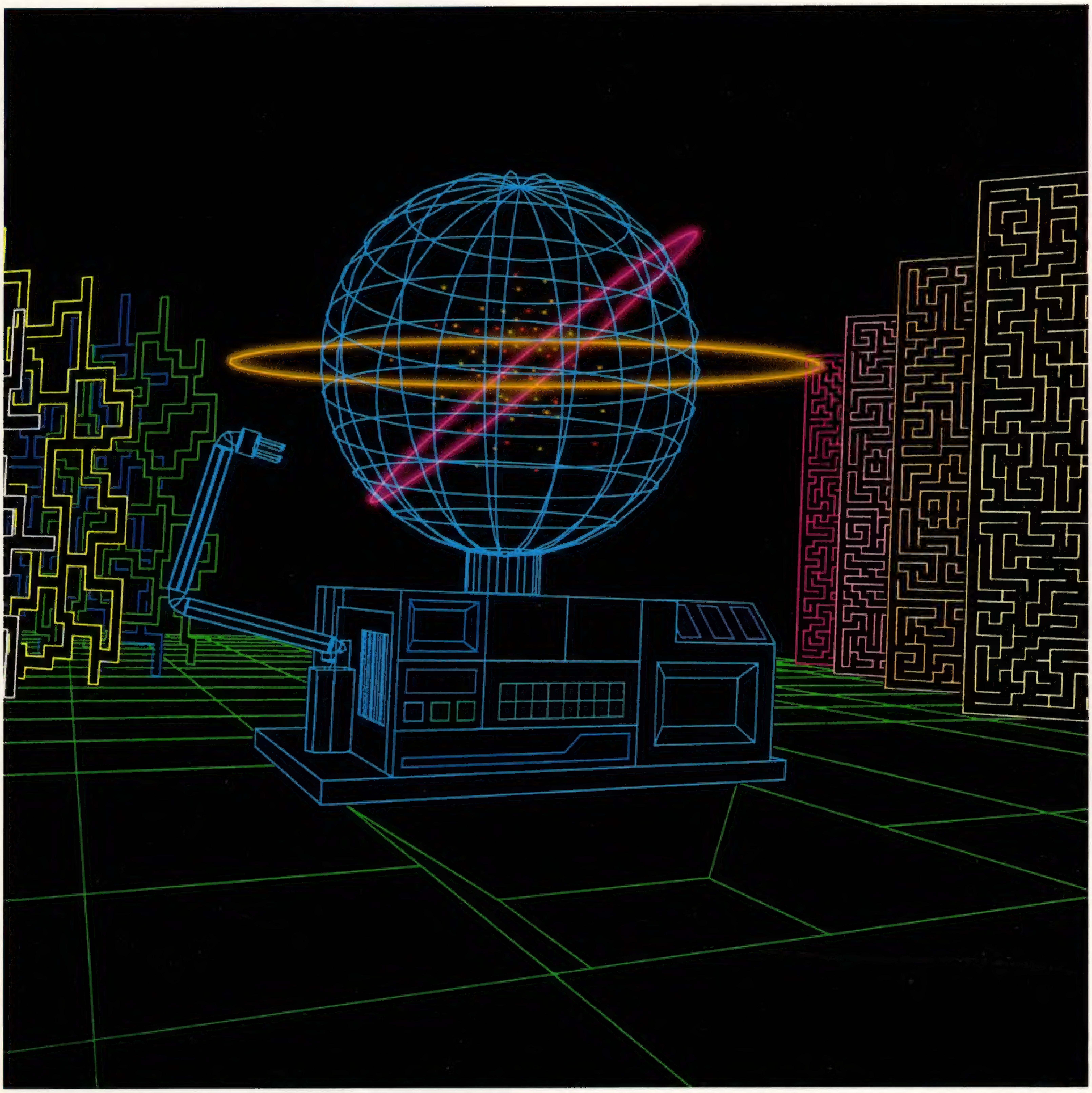
SYSTEM'S BANK

PC88♯3

アスキー・システム・バンク（PC88♯3）

PC-8801マシン語入門

藤井敬雄・民田雅人共著



アスキー出版局

Computer Graphics Design by Masaru Kabutoya
Computer Graphics Operation by Tatsuo Nagamatsu
Photo by Yutaka Yagai／StudioV

PC-8801　マシン語入門

藤井敬雄　民田雅人　共著

アスキー出版局

本書を読む前に

本書は，NEC PC-8801をベースにしてマシン語を学習しようと考えている方のために書かれたものです．

マシン語の習得をめざす方の目的は様々なものがあると思いますが，読者の方々がどのような目的をもって読むにしても，マシン語を学習する上で必要な基本的事項について詳細に解説してあります．

従来出ていた多くのマシン語の解説書は，特定の機種に限定されることはありませんでした．なぜならPC-8801にしても，NECのCPUμPD780C-1（Z80Aコンパチブル）を搭載していますから，CPUの命令セットについては数多くの解説書がフォローしています．しかし，特定の機種でマシン語プログラミングを行い，そのグラフィック機能，I/Oポート，インタプリタの力をフルに引き出すためには，CPUの機能特性のみでなくシステム全体の特性を考慮することが必要です．本書は上記の意図に則って，PC-8801の機能に準じたプログラムを中心に構成されています．

ユーザが本書を通読することによってマシン語を学び，PC-8801の性能を充分に活用することを望んでやみません．

次に本書の構成について紹介しておきましょう．

第1章　マシン語とは何か

マシン語とBASICインタプリタやコンパイラとの比較によって，マシン語の長所・短所を浮き彫りにしています．マシン語の実体を知るとともに，マシン語の本当の価値を説明します．

第2章　マイコン基礎知識

マシン語でプログラムを作成する上で必要なマイコンの基礎知識を解説します．2進数，16進数，10進数といったn進数の話に始まり，キャラクタ・コードやハードウェアの基本的な用語の解説を行っています．すでにBASICのプログラミングの経験がある方も，再度確認する意味で一読されることをお勧めします．

第3章　マシン語の基礎知識

マシン語の各種命令の説明に入る前に，マシン語の学習に必要不可欠な用語の意味や様々なマシン語ツールについて解説します．

第4章　モニタとアドレス

N_{88}-BASICモードのモニタの使用説明とメモリ構成，マシン語領域の確保といった，マシン語の編集機能について述べています．N_{88}-BASICのモニタ機能はCP／M*のDDT(Dynamic Debugging Tool)に準じたものですから，マシン語プログラムの開発に大きな助力となることでしょう．

第5章　マシン語プログラミング

各種命令を組み合わせたサンプル・プログラムを例にして，命令の実際の使用法を解説します．すべての命令を解説することは，初心者の方には混乱を招くことになるので，頻繁に使用されるものを説明しています．この章の命令を理解した後には，マシン語のプログラムを解析することが可能になるでしょう．また〝Appendix〟に掲載されているGlibのソース・リストには，解析の助けとなるようにコメントを付けておきましたので，独自に解析すれば確実に力がつくはずです．また，本章より学習ツールとしてMF-ASM(エムエフ・アセンブラ)を利用しますので，ここで簡単な使用法を説明します．

第6章　マシン語プログラム・テクニック

この章では，前章までに学んだ知識をもとに，いよいよ本格的なプログラミングへの扉をたたいてみましょう．マシン語プログラミングの心得に始まり，BASICとのリンク，検索，データの変換，I／Oポートの利用，グラフィックス，ランダム・テクニック等，アルゴリズムを中心にプログラムの解析を行います．

第7章　開発ソフトウェア

現在PC-8801で利用可能なマシン語開発ツールの紹介を行います．１つは，DUAD-88D(ASCII製)というアセンブラ，ディスアセンブラ，デバッガ，ローダを含むユーティリティ・ソフトウェアであり，もう１つはCP／Mという汎用OSです．これらは今すぐ必要なツールではありませんが，効果的なマシン語プログラミングの開発にはなくてはならないものです．

Appendix

付録として２つのマシン語ユーティリティのプログラムを掲載し，説明を行っています．

(1)　MF-ASM（エムエフ・アセンブラ）

(2)　Glib（グラフィック・ライブラリ）（『PC-8801 グラフィックスのすべて』より引用）

この2つのマシン語プログラムの説明は，今までの学習の集大成の意味で付け加えました．読者の皆さんが独自に解析される際の参考になると思います．

また，MF-ASMはPC-8801用のザイログ文法準拠のアセンブラであり，ある程度実用に堪えるプログラムとして作られています．

従来のマシン語の学習は，主にハンドアセンブルを前提として考えられていました．しかし，ソフトウェアの習得という立場から考えた場合，ハンドアセンブルという作業は決して合理的な作業とはいえません．

本書では，読者の皆さんがMF-ASMを利用することによって，マシン語の学習やプログラムの記述をより効果的に行うことを目的としています．

本書のプログラムは，全体を通して命令の理解とアルゴリズムのわかりやすさを中心に考えていますので，必ずしもプログラムとしては最適化されているものではありません．また，本文中のプログラムを実際に入力する場合には，くれぐれもタイプミスのないよう注意すると共に，Gコマンドを実行する前にセーブすることをお勧めします．

本書で用いた標準システム

本書の中で標準システムとしているのは，本体＋カラーCRT(解像度640×200)＋5インチディスク・ユニットですが，外部記憶装置にカセットテープを利用している方も使うことができます．

＊ CP/M®は米デジタルリサーチ社の登録商標です．

出版に当たって

本書の出版に当たり，株式会社システムソフト福岡の御協力を得ております．『PC-Techknow8800 Vol.1』よりの引用を心よく承諾いただきましたことを，ここに厚く御礼申し上げます．

目　次

本書を読む前に……………………………………………………………………3

1章マシン語とは何か………13

1.1———マシン語の姿………14
1.2———BASICはなぜ動くか………15
  1.2.1——BASICプログラムを動かすもの………15
  1.2.2——インタプリタとコンパイラ…………16
1.3———マシン語の長所・短所………17
  1.3.1——マシン語の長所………17
  1.3.2——マシン語の短所………17
  1.3.3——結論………18
1.4———マシン語を利用するには………18

2章マイコン基礎知識………19

2.1———マシン語と２進数………19
  2.1.1——２進数の用語………21
2.2———２進数と16進数………22
  2.2.1——16進数のきまり………24
2.3———２進↔10進相互変換………25
  2.3.1——２進→10進変換………25
  2.3.2——10進→２進変換………26
2.4———２進数の演算　(1)………26
  2.4.1——加算………27
  2.4.2——減算………27
  2.4.3——補数………27

2.5——2進数の演算 (2)………29
  2.5.1——乗算………29
  2.5.2——除算………30
  2.5.3——シフト演算………31
2.6——BCDコードとは………32
  2.6.1——BCDの特徴………33
  2.6.2——BCDの計算………33
2.7——論理演算………34
  2.7.1——論理和(OR)………35
  2.7.2——論理積(AND)………36
  2.7.3——否定(NOT)………37
  2.7.4——論理演算の値………37
  2.7.5——排他的論理和………38
  2.7.6——論理演算の利用法………39
2.8——プログラミングのためのハードウェア知識………40
  2.8.1——CPU………41
  2.8.2——メモリ(記憶装置)………41
  2.8.3——アドレス(番地)………42
  2.8.4——I／O(入／出力装置)………42
  2.8.5——I／Oポート………42
  2.8.6——レジスタ………42
  2.8.7——情報の流れ………43
2.9——符号による表現………43

3章マシン語基礎知識………47

3.1——CPUとマシン語………47
  3.1.1——CPUのいろいろ………47
3.2——μPD780のレジスタ構成………48
  3.2.1——主レジスタと副レジスタ………49
  3.2.2——アキュムレータ………50
  3.2.3——汎用レジスタ………50
  3.2.4——インデックス・レジスタ………50
  3.2.5——スタック・ポインタ………50

3.2.6——プログラム・カウンタ………50
3.3——Fレジスタの役割………51
3.3.1——CYフラグ………52
3.3.2——Zフラグ………53
3.3.3——Sフラグ………53
3.3.4——P／Vフラグ………53
3.3.5——Hフラグ，Nフラグ………55
3.4——マシン語とニーモニック………55
3.5——アセンブラと逆アセンブラ………56
3.6——命令長による命令の区別………57
3.7——μPD780の命令………58
3.8——アルゴリズムとフローチャート………59
3.9——サブルーチンについて………60

4章モニタ機能とアドレス………63

4.1——モニタとは………63
4.1.1——モニタの機能………64
4.2——N_{88}-BASICのモニタ………64
4.2.1——モニタで使用する文字………64
4.2.2——コマンドの説明………65
4.3——メモリマップとメモリの使用法………75
4.3.1——危険地帯………75
4.3.2——メモリマップ………76
4.3.3——マシン語領域の確保………78
4.3.4——MAXエリア………80

5章マシン語プログラミング………83

5.1——ニーモニックの表現形式………83
5.1.1——シンボルのはなし………84
5.1.2——μPD780にないニーモニック………86
5.2——マシン語のツール………89

5.3——メモリの内容交換 (1)………90
5.3.1——LD命令………91
5.3.2——メモリの入れ換え………93
5.3.3——チャレンジ………93
5.4——メモリの内容交換 (2)………96
5.4.1——EX命令………96
5.4.2——リストの実行にあたって………97
5.5——メモリの内容交換 (3)………97
5.5.1——INC命令とDEC命令………98
5.5.2——JR命令………99
5.5.3——DJNZ命令………99
5.5.4——ラインの入れ換え………100
5.5.5——チャレンジ………101
5.6——キャラクタで線を引く………102
5.6.1——ADD命令とADC命令(8ビット)………103
5.7——キャラクタ・スクウェア………104
5.7.1——枠を描く………104
5.7.2——SUB命令とSBC命令(8ビット)………105
5.7.3——サブルーチンとCALL命令とRET命令………107
5.7.4——16ビット加／減算命令とCYフラグの操作………110
5.7.5——プログラムバリエーション………112
5.8——G-VRAMの接続を調べる………112
5.8.1——ポートのはなし………113
5.8.2——IN／OUT命令とBIT命令………114
5.8.3——OR命令………115
5.8.4——DI命令とEI命令………115
5.8.5——G-VRAMの選択………115
5.8.6——メモリ・ロケーション………116
5.9——グラフィック画面のイニシャライズ………116
5.9.1——ポートの役割……118
5.9.2——XOR命令………119
5.9.3——LDIR命令………119
5.9.4——CP命令………120
5.10——画面に模様を描く………121
5.10.1——RRCA命令………124

5.10.2——PUSH命令とPOP命令………124
5.10.3——点を打つ………125
5.10.4——JP命令………126
5.11——ハンドアセンブルの方法………127

6章マシン語プログラム・テクニック………131

6.1——危険なマシン語!?………131
6.1.1——マシン語プログラムのデバッグ………132
6.2——BASICプログラムとのリンク………133
6.2.1——USR関数の使い方………133
6.2.2——CALL文の使い方………137
6.3——内部ルーチンの利用………140
6.3.1——内部ルーチンとは………140
6.3.2——内部ルーチンの使い方………141
6.4——データの型と変換………149
6.4.1——1バイトの表すもの………149
6.4.2——データ変換 (1)………149
6.4.3——データ変換 (2)………156
6.5——I／Oポートを使う…………163
6.5.1——キースキャン………163
6.5.2——プリンタ出力………167
6.5.3——テキストを読む………170
6.6——文字列のサーチ………174
6.7——グラフィック・パターンを動かす………177
6.7.1——点を打つ………177
6.7.2——パターンを表示する………178
6.7.3——パターンを動かす………180
6.7.4——シンボル用のデータの作成………186
6.8——内部ルーチンの解析………187
6.8.1——解析のKnow-how………187
6.8.2——解析の実際………188
6.9——ランダム・テクニック………191
6.9.1——乱数の発生………191

6.9.2——メモリの節約………193
6.9.3——スピードについて………195
6.9.4——割込みについて………197

7章 開発ソフトウェアについて………203

7.1———マシン語プログラム開発ソフト………203
7.2———DUAD-88D………203
7.3———多言語システム……CP／Mについて………207

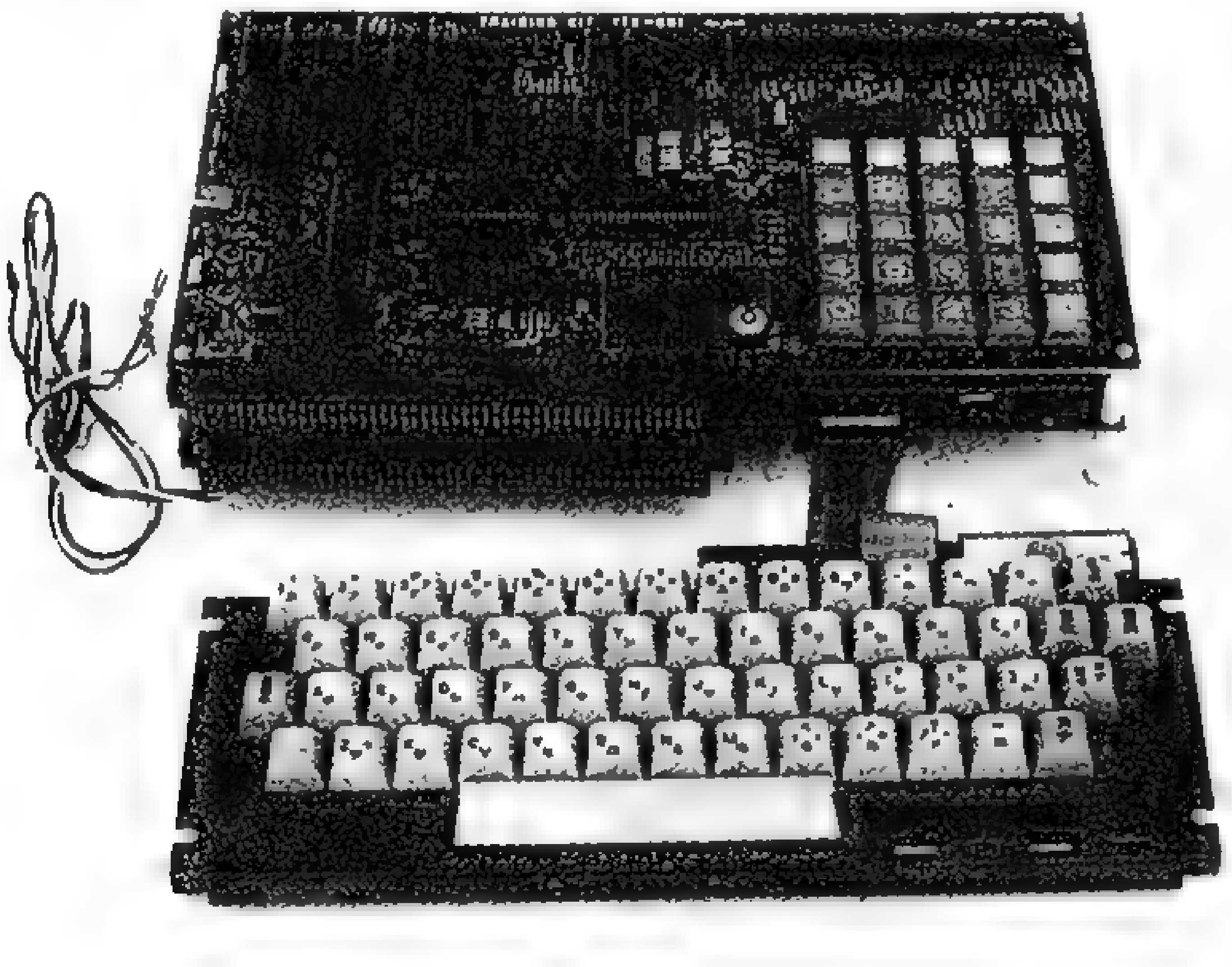
APPENDIX ………209
A.1 MF-ASM………209
A.2 MF-ASM文法………222
A.3 ユーティリティ・ソフトウェア………228
A.4 Gilbソース・プログラムの解析………232
A.5 インストラクション表の見方………251

索引………256

付録・インストラクション表

第1章　マシン語とは何か

最近のパーソナル・コンピュータのBASIC機能の豊富さには目をみはるものがあります．皆さんは余りご存知ないかと思いますが，数年前まではNECがTK-80BSというマイクロコンピュータ用拡張ボードを販売していました．BS(TK-80BS)は，TK-80もしくはTK-80Eというワンボード・マイクロコンピュータ（１枚の基板上に必要な機能を集約したマイコン）と組み合わせることによってBASICのプログラムを実行することができました．しかし，このBASICにはグラフィック機能などはいっさいサポートされておらず，かろうじてN_{88}-BASICのLOCATE文に相当する命令があった程度でした．また，TK-80本体だけで実行することのできる言語はマシン語だけで，BASICのプログラムは実行できなかったのです．ですから少し前のマイコンユーザには，マシン語を最初に学び，その後でBASICを知った人が多い程です．



往年の名機TK-80

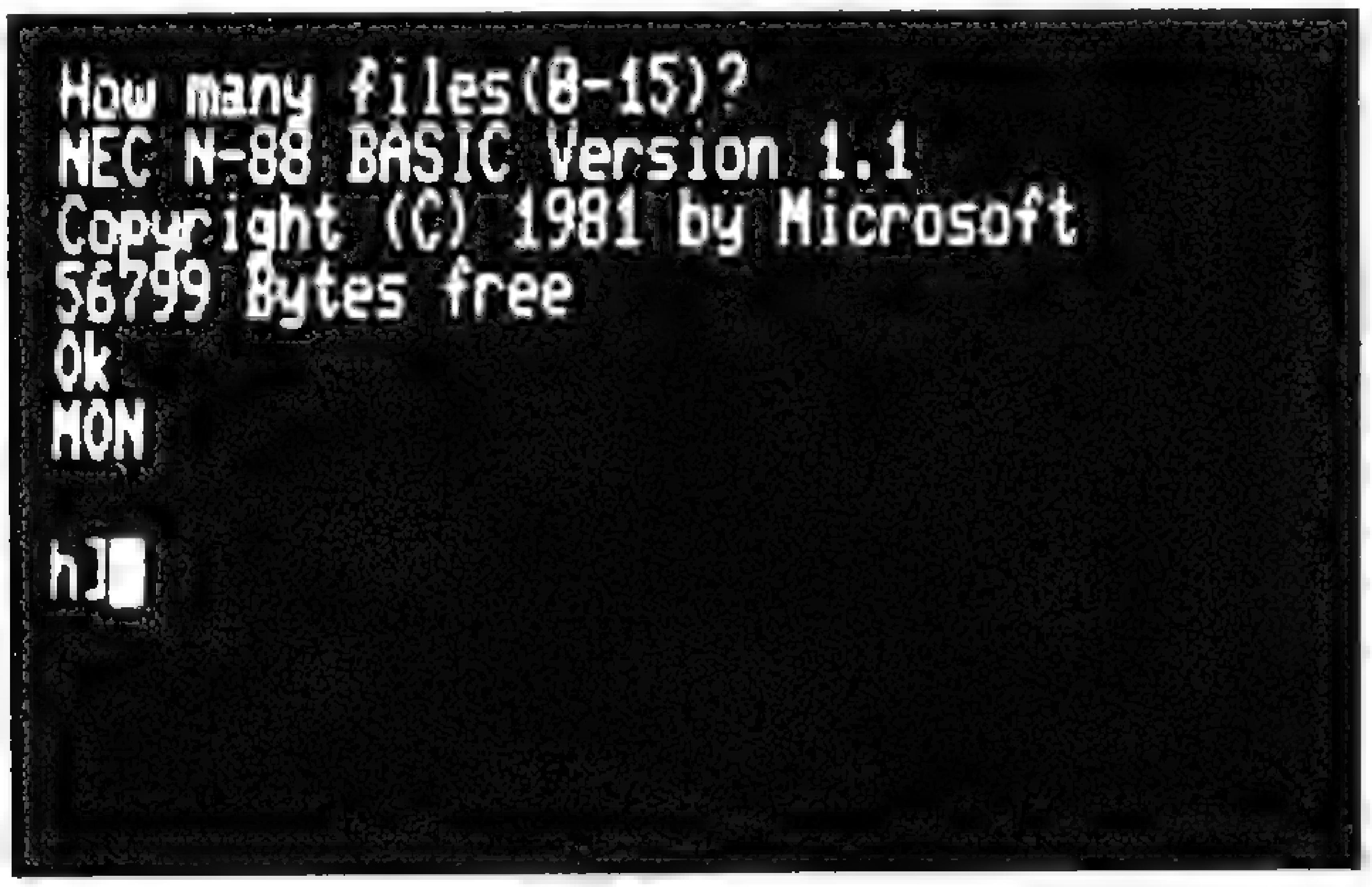
それでは，ワンボードマイコンでも実行できるマシン語とはどんなプログラム言語なのでしょうか？マシン語でプログラムを行うことは，CPUに対して動作の指令を出すことですからBASICとは比較することができない程原始的なものです．ここで言う〝原始的〞とは，人間に理解しにくいということであり，その意味ではBASICも，私達が日常使っている言語に較べれば原始的であるといえます．

さて，この章ではマシン語とはどんな言語であるかその雰囲気をつかむと共に，BASICインタ

プリタ，コンパイラについて説明を行い，マシン語とBASICの違いを明らかにしたいと思います．その上で，なぜマシン語プログラムが必要であるかを理解してください．

1.1 マシン語の姿

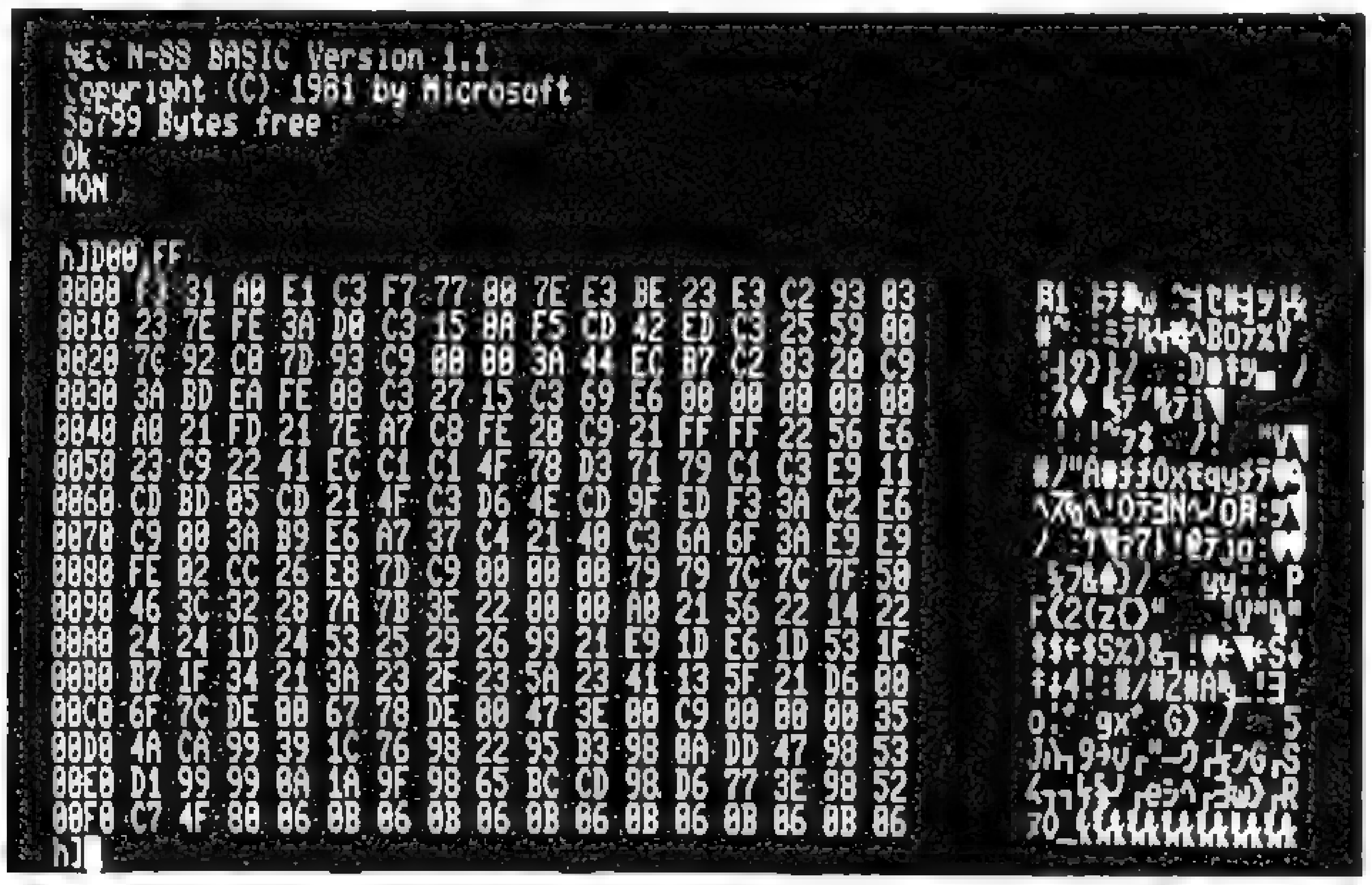
まず，相手の正体をみてみましょう．PC-8801を起動させ，MON↵（↵はリターンキーを押す意）を実行してください．



```
How many files(0-15)?
NEC N-88 BASIC Version 1.1
Copyright (C) 1981 by Microsoft
56799 Bytes free
Ok
MON

h]
```

モニタ・モードへ入ったときの画面

上の写真のようにh]が表示され，隣にカーソルが点滅して，キー入力待ちとなっていると思います．次に，D00,FF↵と入力してください．画面には，何やら数字とアルファベットの集まりが表示されます．



```
NEC N-88 BASIC Version 1.1
Copyright (C) 1981 by Microsoft
56799 Bytes free
Ok
MON

h]D00,FF
0000 F3 31 A0 E1 C3 F7 77 00 7E E3 BE 23 E3 C2 93 03
0010 23 7E FE 3A D0 C3 15 8A F5 CD 42 ED C3 25 59 00
0020 7C 92 C8 7D 93 C9 00 00 3A 44 EC B7 C2 83 20 C9
0030 3A BD EA FE 08 C3 27 15 C3 69 E6 00 00 00 00 00
0040 A0 21 FD 21 7E A7 C8 FE 20 C9 21 FF FF 22 56 E6
0050 23 C9 22 41 EC C1 C1 4F 78 D3 71 79 C1 C3 E9 11
0060 CD BD 05 CD 21 4F C3 D6 4E CD 9F ED F3 3A C2 E6
0070 C9 00 3A B9 E6 A7 37 C4 21 40 C3 6A 6F 3A E9 E9
0080 FE 02 CC 26 E8 7D C9 00 00 00 79 79 7C 7C 7F 50
0090 46 3C 32 28 7A 7B 3E 22 00 00 A0 21 56 22 14 22
00A0 24 24 1D 24 53 25 29 26 99 21 E9 1D E6 1D 53 1F
00B0 B7 1F 34 21 3A 23 2F 23 5A 23 41 13 5F 21 D6 00
00C0 6F 7C DE 00 67 78 DE 00 47 3E 00 C9 00 00 00 35
00D0 4A CA 99 39 1C 76 98 22 95 B3 98 0A DD 47 98 53
00E0 D1 99 99 0A 1A 9F 98 65 BC CD 98 D6 77 3E 98 52
00F0 C7 4F 00 06 0B 06 0B 06 0B 06 0B 06 0B 06 0B 06
h]
```

0000H番地から00FFH番地のメモリ・ダンプ

この場でここに表示された英数字の意味を理解しろというわけではないので，どうぞご安心ください．ここに表示された英数字の並びのうち，2桁ずつで表示されているもの，これがマシン語です．何やら神秘的に感じませんか？しかし恐れるに足りません．ここでは，マシン語の表現法の1つにこのようなものがあることを覚えていただければ結構です．

1.2　BASICはなぜ動くか

話は少し横道にそれますが，皆さんはBASICのプログラムが実行できることを不思議に思ったことはありませんか？現在市販されている多くのパソコンは，電源を入れればBASICを実行することができます．ですからBASICでプログラムを記述している限り，なぜ実行できるかということは考える必要はありません．しかし，これからマシン語を学んでいく上で，BASICとの違いを考えることは決して無駄ではありませんので，ここで少々触れてみることにします．

本章の初めで，TK-80というマイコンは，本体のみではマシン語しか実行できないのにBSという拡張ボードを組み合わせるとBASICプログラムが実行できると述べました．ですから，BSがBASICプログラムを動かしている何かであることは想像がつきます．また，PC-8801でもBASICプログラムが実行できるのですから，当然BSに相当するものがあるはずです．

それでは，BASICプログラムを動かすものとは一体何なのでしょうか．

1.2.1　BASICプログラムを動かすもの

BASICのプログラムを動かすための手段として，一般に2つの方法があります．1つはインタプリタによるもので，もう1つはコンパイラによるものです．

インタプリタとは，BASICの各命令に対応する実行処理ルーチンの集まったプログラムです．インタプリタはBASICプログラムの意味を1行1行解析すると，個々のBASICの命令に対応するルーチンにおいて，その処理が実行されます．つまり，インタプリタからみた場合，BASICのプログラムはインタプリタに一定の処理を行わせるデータと考えることができるのです．

コンパイラとは，BASICプログラムの意味を解析して，個々の命令の意味に対応するマシン語を出力するプログラムです．コンパイラ自身はBASIC→マシン語の変換作業を行うだけであり，目的とするプログラムの結果を得るためにはマシン語に変換されたプログラムを実行しなければなりません．

インタプリタ，コンパイラとも，マシン語でプログラミングされており，BASICのプログラムは，インタプリタなりコンパイラなりがなければ画面に表示された単なる文字に過ぎません．BASICは，インタプリタ，コンパイラによってそれぞれ処理形態は異なりますが，最終的にはマシン語の形式で実行されています．このことは何を意味するかといえば，コンピュータでの実行形態はどのような場合にも最終的にマシン語で処理されているために，マシン語でプログラムを記述すれば，BASICで記述したプログラムを実行する場合に必要な変換処理が不用になるということです．ちなみに，先のBSとN_{88}-BASICはインタプリタ形式をとっています．

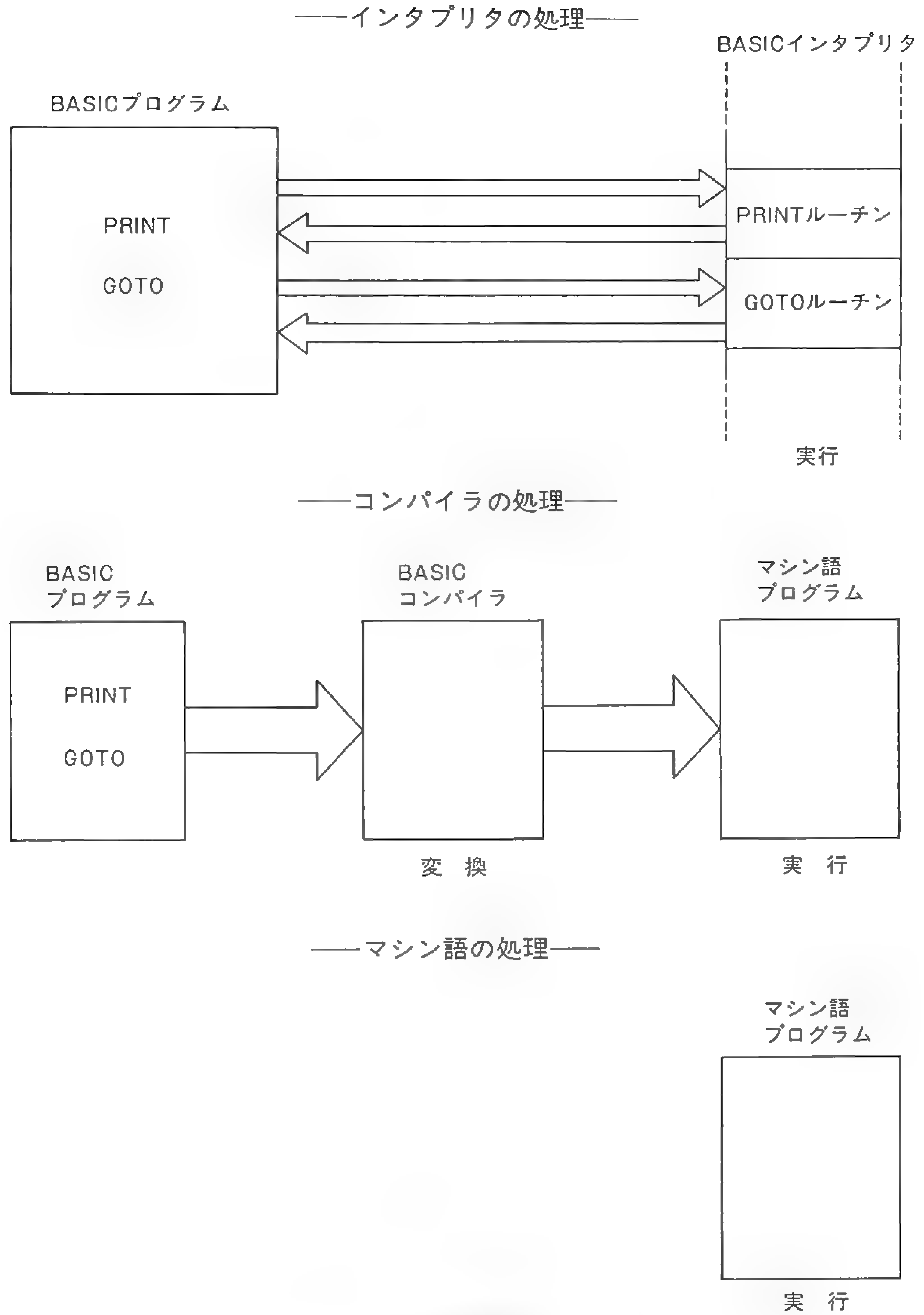

——インタプリタの処理——
BASICインタプリタ
BASICプログラム
PRINT
GOTO
PRINTルーチン
GOTOルーチン
実行
——コンパイラの処理——
BASIC
プログラム
BASIC
コンパイラ
マシン語
プログラム
PRINT
GOTO
変　換
実　行
——マシン語の処理——
マシン語
プログラム
実　行

図1.1　処理系の違い

1.2.2　インタプリタとコンパイラ

インタプリタとコンパイラを比較すると，それぞれ以下のような特徴があります．

インタプリタ	コンパイラ
・実行速度が遅い ・デバッグが容易である ・比較的小規模のシステムで実現できる	・実行速度が速い ・デバッグに手間がかかる ・比較的大規模のシステムでないと実現できない

表1.1　インタプリタとコンパイラの特徴

なお，ここでいうデバッグとは，文法誤りではなく算法誤りです．文法誤りについては，どちらも容易に発見することができます．

1.3　マシン語の長所・短所

マシン語とBASICの処理形態の違いが判明したところで，マシン語の長所と短所についてBASICと比較しながら説明しましょう．

1.3.1　マシン語の長所

マシン語を利用するメリットは以下の点に集約されます．

■ スピード

これは，マシン語のもつ最大の長所といっても過言ではありません．一般のBASICインタプリタと比較すると，その実行スピードは数百倍から，時には数万倍に及ぶことがあります．これは先程述べたインタプリタがBASICのプログラムを１行１行解釈実行を繰り返すのに対し，マシン語はCPUが直接実行できるからです．

■ メモリの使用効率

これもマシン語の大きな特徴の１つです．同じプログラムをBASICで組んだものと比較すると，かなり少ない容量のメモリで実現することが可能になります．もっとも，最近はメモリコストが下がったため，この点はあまり問題となりません．しかし，マシン語のプログラムだけを扱う場合には，PC-8801のメモリが足りなくなるということは，まず起こらないでしょう．

1.3.2　マシン語の短所

さて，上記のような長所があるのにどうしてマシン語が一般に広く利用されないかというと，それは次のような大きな短所があるからです．

■ 手続きが細かい

これはマシン語最大の短所です．例えば，１＋１の答えを表示させようとするなら，BASICでは

```
PRINT　1＋1↵
```

と，たった１行で，しかもひとめ見ただけで何を行うのかわかります．しかしマシン語ではそうはいきません．これだけのことを行うにも数個から数十個の命令を組み合わせなければなりませんし，また1.1で示したように，ひとめ見ただけではその意味するところが理解できないのです．

■ プログラミングが煩雑である

これは前述したことにつながりますが，何を行うにも，BASICと比較するとかなりの手間がかかります．

■ デバッグに時間がかかる

これは，前に述べた2つのことから起こり得る当然のことといえるでしょう．

1.3.3　結論

以上で述べてきたように，マシン語でプログラムを作るにはたいへんな労力と時間を要するのです．それならばコンパイラを利用すればよいと思われるかもしれませんが，コンパイラを使用して生成されたマシン語は，直接作成したマシン語プログラムと比較すると実行速度も遅く，メモリも余計に消費します．これは，コンパイラが1つの命令に対してある程度決まったパターンのマシン語にしか変換できないため，マシン語としてみた場合かなり無駄が多く，結果としてスピードの低下，メモリの使用効率の低下を招くからです．

1.4　マシン語を利用するには

前項でマシン語の長所および短所を理解していただけたことと思います．ですから，マシン語を利用するためには，前項で述べたようなことをよく考えて，メリットがデメリットを上回った場合に利用することです．まず考えられるのは，マシン語のもつ実行スピードを最大限に活かすことでしょう．ゲーム･プログラムなどで「どうしてもBASICでは，実行速度が遅くて実用に堪えない」といった場合には，やはりマシン語を使用するのが適しています．しかしBASICで実行すればかなり時間がかかるようなものとして，素数を求めるプログラムを作ったとします．これなどはBASICを利用すれば，プログラムの組み方によっては数十秒(BASICインタプリタの中でも最高速に類するものを使用した場合)から数十分，時には1時間を超えることもあります．マシン語を利用すればせいぜい数秒で終わります．確かに速いことはよいことなのですが，素数を求めるプログラムは一度計算してプリンタに出力してしまえば終わりですから，マシン語のプログラム開発に数週間を要していたのではお話になりません．

前項で述べた通り，マシン語のプログラムを作るためには相当の労力と時間を要しますので，そのことはよく考慮すべきでしょう．

第2章　マイコン基礎知識

マシン語がよくわからないといわれる理由の１つとして，マイコンで使われる用語に不可解なものが多いことが挙げられると思います．そこで，この章ではマシン語を学ぶ上でしばしば使われる用語について解説すると共に，コンピュータにまつわる各種演算（２進，16進，論理演算）の方法についても紹介していきます．

BASICのプログラミングの経験のある方にとっては，すでにご存知の事項もあるとは思いますが，知識を確認する意味でも一読されることをお勧めします．

2.1　マシン語と2進数

私たちが普段数を数える場合には，０～９の文字を使用した10進法を用います．そして数を識別する場合は，０～９の文字の形を判断し，これらの文字列の並びが表す意味を読み取ります．ところで，コンピュータが数を数えたり，識別したりするのには何を用いているのでしょうか．私たちが使用する０から９までの数字に相当するものとして，コンピュータは電圧を使用しています．つまり，電圧の違いを数字の違いとして認識しているのです．具体的には，コンピュータは基準になる電圧を用意しており，その基準電圧より高い電圧か低い電圧かを判断しているのです．例えば，０～10Ｖまでの電圧を扱うことのできるコンピュータを考えてみましょう．このコンピュータには基準電圧として1,2,3,4,5,6,7,8,9Ｖの各電圧が必要であり，０～１Ｖの範囲にある電圧を０，１～２Ｖの範囲にある電圧を１といった具合に数を識別するので，10進法を扱うことができるようになります（図2.1参照）．

しかし，実際にはこのようなコンピュータは存在しません．９個もの基準電圧を用意して電圧を識別するには，ハードウェアが複雑になり，またどうしてもノイズなどに左右されやすく，誤動作の原因になります．

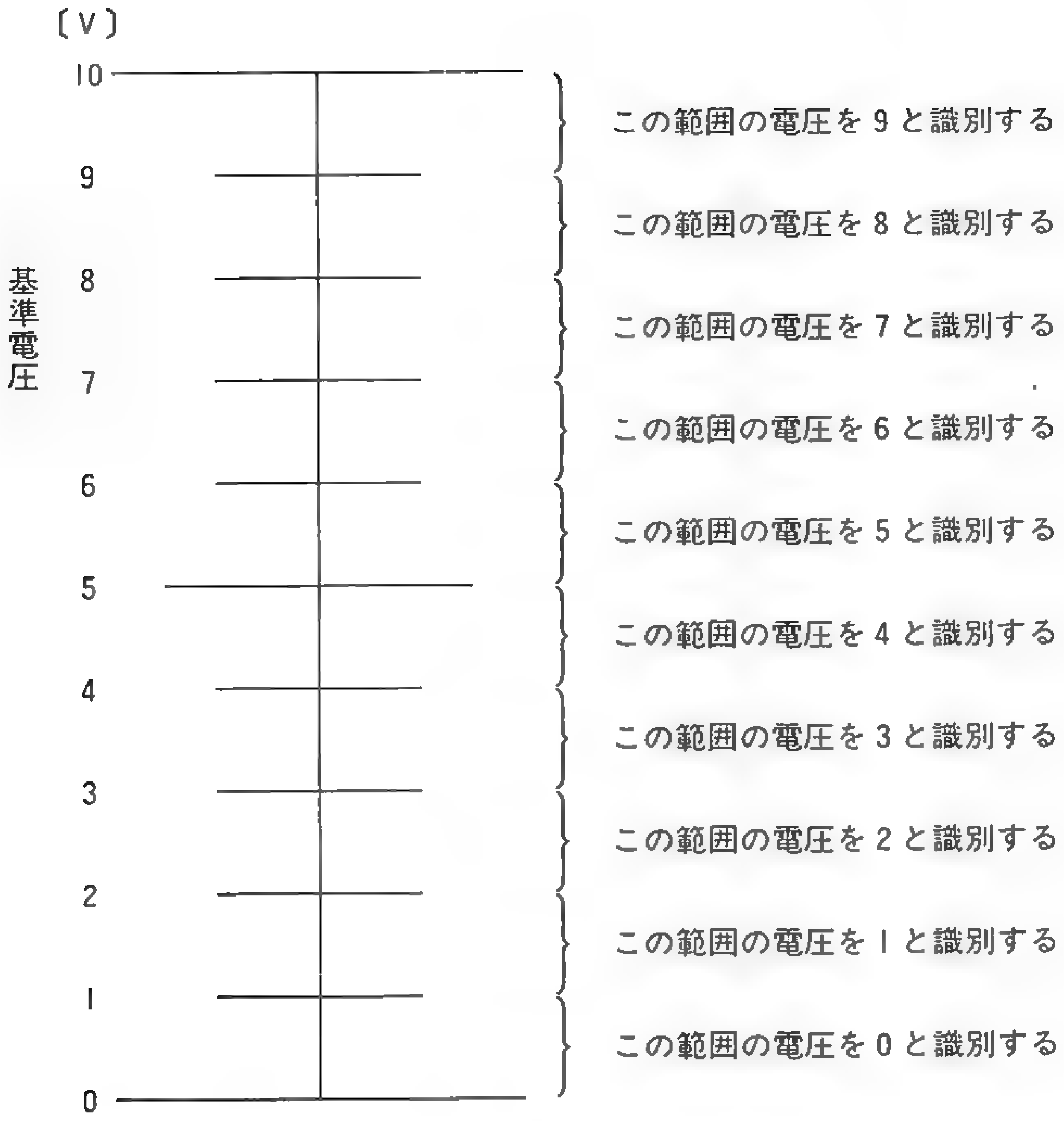
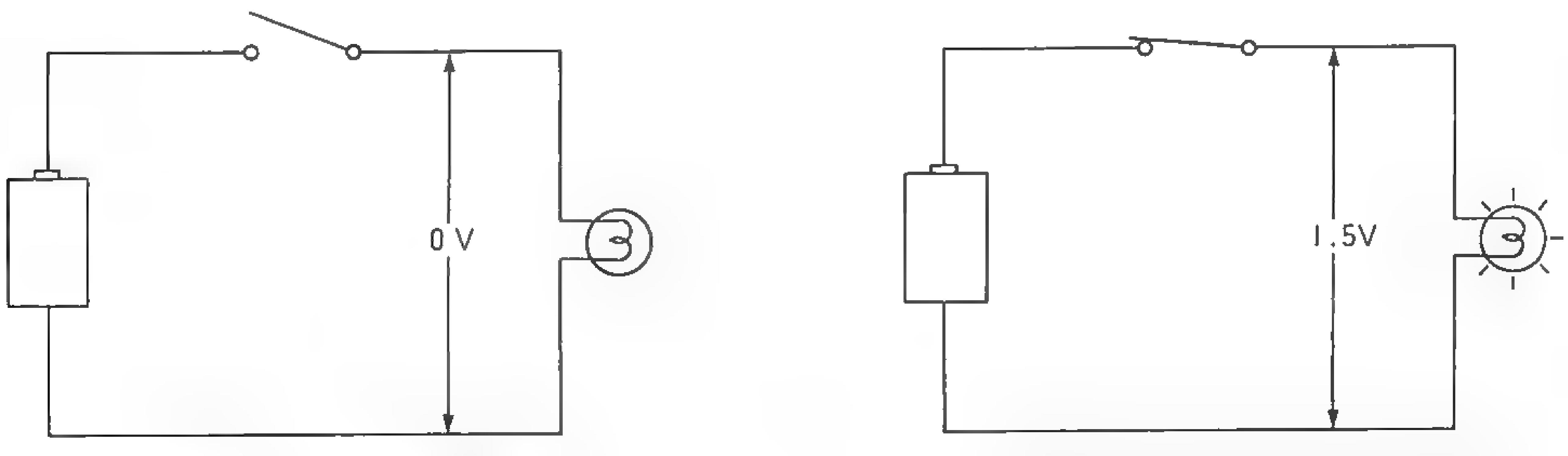

〔V〕
10
9
8
7
6
5
4
3
2
1
0
基準電圧
この範囲の電圧を9と識別する
この範囲の電圧を8と識別する
この範囲の電圧を7と識別する
この範囲の電圧を6と識別する
この範囲の電圧を5と識別する
この範囲の電圧を4と識別する
この範囲の電圧を3と識別する
この範囲の電圧を2と識別する
この範囲の電圧を1と識別する
この範囲の電圧を0と識別する

図2.1　電圧と数字

それでは，このように電圧を使用する場合にはどうすればよいのでしょうか．単純な解決法としては，基準電圧を減らして，扱うことのできる数の種類を少なくすることが考えられます．究極的には，基準電圧が1個，扱うことのできる数は2種類というのが最も合理的であるといえるでしょう．つまり10進法ではなく2進法を用いることによって，扱う数が2種類(0，1)になりますから，コンピュータを設計する上で回路が簡単になり，信頼性も向上しますし，計算などもとても処理しやすくなります．



スイッチが開いている状態＝0に相当する　　スイッチが閉じている状態＝1に相当する

図2.2　0，1の状態

例えば，加算を例にとって考えてみましょう．10進数で1桁の加算というのは，次頁に示すように55種類ありますが，2進数では3種類しかありません．

$$\text{10進数}\left\{\begin{array}{llll} 0+0=0 & 0+1=1 & 0+2=2 & \cdots\cdots \\ 1+1=2 & 1+2=3 & 1+3=4 & \cdots\cdots \\ 2+2=4 & 2+3=5 & 2+4=6 & \cdots\cdots \\ 3+3=6 & 3+4=7 & 3+5=8 & \cdots\cdots \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \end{array}\right.$$

$$\text{2進数}\left\{\begin{array}{l} 0+0=0 \\ 1+0=0+1=1 \\ 1+1=10 \end{array}\right.$$

表2.1　10進数と2進数の1桁加算

これだけを例にとっても，いかに2進数というものが単純であるかということがおわかりいただけたと思います．つまり先に述べたハードウェアの制約を考えて，コンピュータが扱わなければならない状態を最低限に留めた結果，10進数よりも2進数の方が相性がよいのです．

2.1.1　2進数の用語

コンピュータで2進数が使われていることがわかったところで，2進数の用語について説明しましょう．

▶ビット

2進数の1桁1桁をビット(Bit)と呼びます．また2進数の1桁は，1と0でしか表すことができないため，例えば$101_{(2)}$なら3ビットの2進数ということになります．皆さんが今使っているマイコンはPC-8801ですが，カタログや説明書をみると，CPUは〝μPD780C-1（Z80Aコンパチブル）8ビット並列処理〟と書かれていると思います．これはCPUが同時に8ビットのデータを扱うことを意味するもので，このようなCPUを8ビットCPUと呼びます．

▶バイト

2進数の8ビットを1バイト(Byte)と呼びます．つまり8ビットCPUは，同時に1バイトのデータを処理できるということです．

▶MSB

2進数の最上位のビットをMSB(Most Significant Bit の略)と呼びます．これは「最も重みのあるビット」という意味です．というのは，10進数の16を2進数で表すと10000となりますが，MSBを間違えて0(2進数では，0と1しかない)とすると00000＝0となってしまうからです．

▶LSB

MSBと反対の意味をもつ言葉が，LSB(Least Significant Bit の略)です．これは，2進数で最下位のビットのことです．

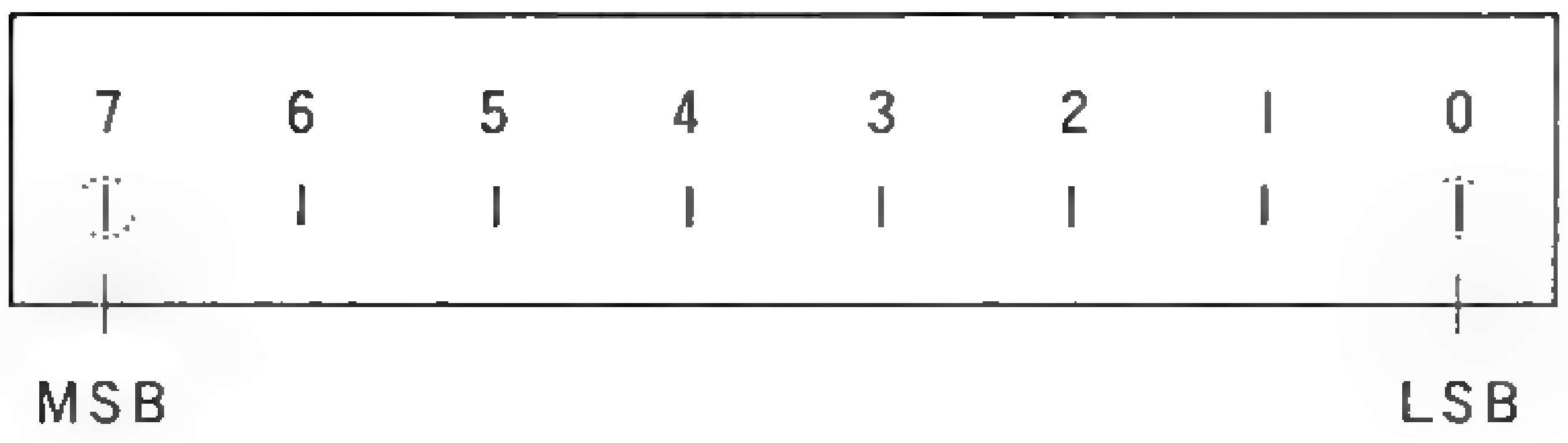


図2.3　MSBとLSB

先程，MSBは「最も重みのあるビット」と述べましたが，マシン語においてはどのビットもたいへん重要な意味をもっています．どのようなCPUであっても，命令のビットを1つ間違えれば正常な動作をしてくれません．正常な動作をしないだけならよいのですが，最悪の場合には暴走してしまい，せっかく苦労して作ったプログラムは破壊されてしまいます．

最後に，今出てきた言葉を簡単にまとめておきましょう．

ビット（Bit）　：2進数の1桁の意
バイト（Byte）：1バイトは8ビット
MSB　：2進数で最上位のビット
LSB　：2進数で最下位のビット

2.2　2進数と16進数

さて，2進数が少しわかったところで，10進数と比較しながら数を数えてみましょう．

10進数	2進数	10進数	2進数
0	0	20	10100
1	1	30	11110
2	10	50	110010
3	11	100	1100100
4	100	500	111110100
5	101	1000	1111101000
6	110	10000	10011100010000
7	111		
8	1000		
9	1001		
10	1010		
11	1011		
12	1100		
13	1101		
14	1110		
15	1111		

表2.2　2進数と10進数の比較

表2.2を参照してください．だんだんビット数が多くなってきて1と0がわかりづらくなってきました．特に10進数で表された10000は，2進数で表現するためには，14ビットも必要になります．これだけのビットを1と0で正確に記述するのは，とてもたいへんなことです．それを，間違いのないように表現する方法として，16進数があります．これは，2進数の4ビットを1桁で表現することができるものです．実際にどうなるのかは，表2.3を参照してください．

10進数	2進数	16進数
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

表2.3　10進数，2進数，16進数 対応表

いかがですか，0～9は10進数も16進数も同じですが10進数の10～15は，16進数ではA～Fで表されます．また，このように4ビットを1つの単位として扱うことから，2進数で0を表現する場合，単に1つの0を書くのではなく，0000と4ビット単位で書くことが習慣となっています．

それでは先程の2進数で14ビットもある数を16進数に直してみましょう．

まず，下の桁から4ビットずつに区切ります．

```
10  0111  0001  0000
```

これを表2.3と照合しながら16進数に変換すると下記のようになります．

```
10  0111  0001  0000
 2     7     1     0
```

つまり10進数で10000という数は，16進数では2710となります．

2.2.1　16進数のきまり

数字を書く場合に単に2710とあると，これは10進数の2710なのか，16進数の2710なのか，全く区別することができません．5A6とか，B6Aなどは，確かに16進数のようですが，16進数が0～9は10進数と同じ文字を使用するため，各桁が0～9の文字ばかりの数は，16進数なのか10進数なのか区別がつかないのです．このようなことを防ぐために，16進数で表す場合には1つの約束があります．それは，**数字の後には，Hを付ける**というものです．Hとは Hexadecimal （16進数の意）の頭の文字をとったものです．

BASICでは，16進数を表現するときには，数字の頭に&Hを付けましたがマシン語を扱う場合には，数字の後にHを付けるということをよく覚えておいてください．ただし，このことは10進数と区別する必要がある場合のみの約束であり，16進数とわかっている場合はHは付けません．

しかし，これだけではまだ不充分なのです．ここで16進数の英文字だけで表されるものを考えてみましょう．例えばBACHという数があったとします．この文字を見て何か思い当たりませんか？そうです，かの大音楽家のバッハになってしまうのです．これは少々極端な例ですが，16進数がアルファベットのA～Fと，10進数と区別するためのHを使うためです．そこで16進数のもう1つの約束として，16進数がA～Fのアルファベットで始まる場合は，頭に0(ゼロ)を付けます．

これで10進数や，英語の単語などからは完全に区別できるようになりました．以下の表記は16進表現の例です．

0H　　1H　　2710H　　5AH　　0FH　　60AH………

2進数で表現される数字は少しでもわかりやすくするために16進数で表現できると述べましたが，これは数値だけにいえることではありません．マシン語の命令も，数値の場合と同じ理由によって，やはり16進数で表されます．命令についての詳細は第5章に譲ることにして，今までに出てきた事項を最後に簡単にまとめておきましょう．

(1)　16進数は，2進数の4ビットを1桁で表すことができる
(2)　16進数を他の10進数と区別する必要がある場合は，最後に必ずHを付ける．また最初がA～Fの場合は，頭に0(ゼロ)を付ける
(3)　マシン語は16進数で表現することができる

ここで皆さんに1つ断わらなければならないことがあります．本書では本文中に出てくる英字で始まる16進数には0を付けていません．「なんだ今いったことを舌の根も乾かないうちに」と思われるでしょう．確かにアセンブラでマシン語を記述する際にはこの規則は厳然として存在するのですが，普通××××H番地という場合にさらに頭に0を付けることは非常にわかりにくくなってしまうので，本書では本文中の16進数に0が付いていなくても，アセンブラを利用する際には付けなければならないことを了解しておいてください．

2.3　2進⟷10進相互変換

本章の１節で述べたように，現在のコンピュータは２進数を使用することを余儀無くされています．私たちが高級言語を使用している限り，日常使い慣れた10進数を使用することができますが，マシン語を使用する場合はコンピュータに歩み寄らなければなりません．つまり，２進数や16進数を扱うことは必然的なことなのです．

２進数や16進数をみて直観的に計算できるという特技をもっている人は別ですが，普通の人にとっては，16進↔10進，２進↔10進変換が必要になりますので，ここでは簡単に２進数，10進数の変換の方法について説明することにします．

変換の方法にはいろいろありますが，ここでは１つの例を挙げるだけに留めておきます．これを参考に，皆さんのわかりやすい方法をマスターしてください．

2.3.1　2進→10進変換

まず，10進数の意味を改めて考えてみることにします．10進数で429の意味は次のように表せます．

$$
\begin{aligned}
429 &= 100\times 4 + 10\times 2 + 1\times 9 \\
&= 10^2\times 4 + 10^1\times 2 + 10^0\times 9
\end{aligned}
$$

数の位の重み

２進数でも同様に考えて，1101という数を例にとります．

$$
\begin{aligned}
1101 &= 2^3\times 1 + 2^2\times 1 + 2^1\times 0 + 2^0\times 1 \\
&= 8\times 1 + 4\times 1 + 2\times 0 + 1\times 1 \\
&= 13
\end{aligned}
$$

この方法ですと，何ビットの数であろうと同じように扱えますから，次のような場合も少々複雑にはなりますがうまくいくようです．

$$
\begin{aligned}
101101100011 &= 2^{11}\times 1 + 2^{10}\times 0 + 2^9\times 1 + 2^8\times 1 + 2^7\times 0 + 2^6\times 1 + 2^5\times 1 \\
&\quad + 2^4\times 0 + 2^3\times 0 + 2^2\times 0 + 2^1\times 1 + 2^0\times 1 \\
&= 2048+512+256+64+32+2+1 \\
&= 2915
\end{aligned}
$$

しかしながら，マイコンがあるのですから，先程の２進数を16進数に変換してみましょう．まず，２進数を16進数に変換します．

1 0 1 1	0 1 1 0	0 0 1 1
B	6	3

次にB63HをBASICのダイレクトステートメントを利用すれば，即座に10進数に変換した数，つまり2915が得られます．

```
PRINT  &HB63  ↵
```

2.3.2　10進→2進変換

それでは今の逆，つまり10進→2進変換を行ってみましょう．これは図2.4のように行います．この例では10進数の123を2進数に変換していますが，どの10進数であっても順々に2で割っていき，最後に得られた商をMSBに，そのときの余りをMSBから2番目のビットに，その前の余りを3番目のビットに，最初の余りをLSBとなるようにすると，その10進数に対応する2進数が得られます．

BASICでは，10進数を直接2進数に変換するコマンドはありませんが，HEX＄を利用すれば16進数の7Bとなり，これを2進数に変換しても，答えを得ることができます．

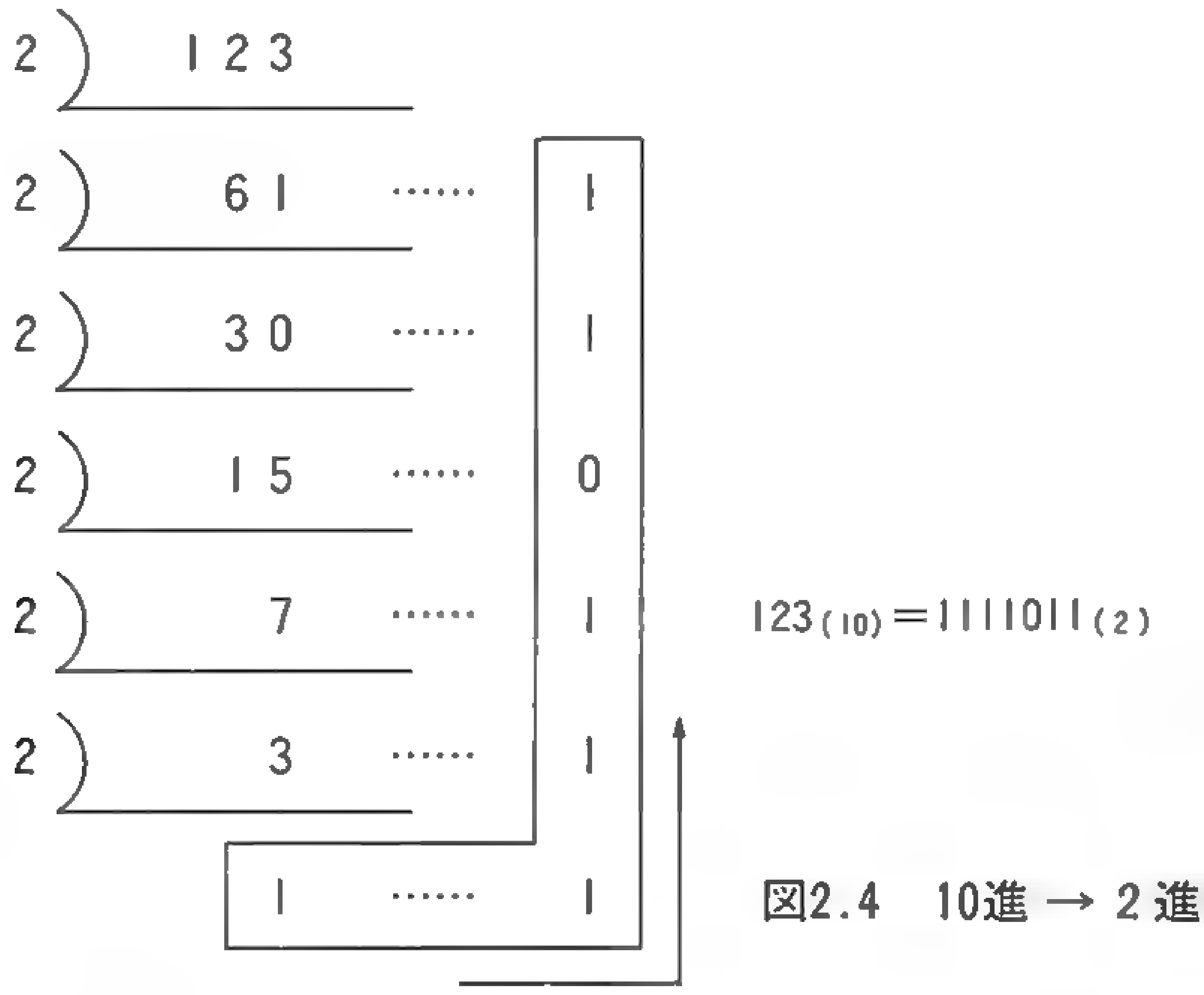


図2.4　10進 → 2進変換の方法

```
PRINT  HEX$(123)  ↵
   7    B
0111 1011
```

2.4　2進数の演算(1)

BASICのプログラムを組んだことのある人ならよくわかると思いますが，プログラムには必ず計算が必要になってきます．このことは，マシン語のプログラムを組む場合にも例外ではありません．ただBASICと違うのは，BASICでは10進数の計算は何の制約もなくできますが，マシン語では2進数か16進数で計算しなければならないことです．

ここでは基本となる2進数の演算について説明します．なお，以下の説明では，CPUが8ビットであることを考えて，2進数は8ビットまたはその半分の4ビットに統一します．

2.4.1　加算

まず加算です．例として10進数の28と37の加算を行ってみましょう．最初に，それぞれの数を２進数に変換します．

$$
\begin{aligned}
28_{(10)} &= 00011100_{(2)} \\
37_{(10)} &= 00100101_{(2)}
\end{aligned}
$$

図2.5　２進数の計算(1)

２進数の加算は，10進数の加算となんら変わりありません．ただ１つ注意する点は，１＋１＝２とはならず，１＋１＝10となって，桁上がりすることです．では実際に行ってみましょう．

$$
\begin{array}{rl}
\quad 111\quad\; & \text{……桁上がり} \\
00011100_{(2)} & \\
+\quad 00100101_{(2)} & \\
\hline
01000001_{(2)} &
\end{array}
$$

図2.6　２進数の計算(2)

2.4.2　減算

次は，減算です．これもやはり10進数の減算と変わりありません．ただ，桁借りが生じたときは，やはり２進数であるということに注意してください．例として10進数の11－５＝６にあたる計算を行ってみます．

$$
\begin{aligned}
11_{(10)} &= 1011_{(2)} \\
5_{(10)} &= 0101_{(2)}
\end{aligned}
\qquad\qquad
\begin{array}{r}
\text{桁借り} \\
1011 \\
-\quad 0101 \\
\hline
0110
\end{array}
$$

図2.7　２進数の計算(3)

答えは$0110_{(2)}=6_{(10)}$となります．

2.4.3　補数

さて，今の減算の例は普段私たちが行うのと全く同じ方法で行いました．ところがコンピュータの内部においてこのような計算方法を用いると，加算回路と減算回路の２種類が必要となり，回路の複雑化やコストの上昇を招きます．しかし，減算を加算回路で行うことができれば減算回路は不要となり，回路も単純になります．この考え方が，補数と呼ばれるものです．これは，次のような数式の変換に似ています．

$$A-B=A+(-B)$$

補数の作り方は，基数をｂとしたとき，与えられた各桁の数字をｂ－１から引き，最下位に１を加えた数を，ｂの補数といいます．例えば，２進数の0110の，２の補数は1010となります（図2.8参照）．

基数２の場合のb-1	1111	
もとの数	－　0110	……(a)
各桁を２－１　つまり１から引いた結果	1001	……(b)
最下位に１を足す	＋　　1	
もとの数の２の補数	1010	……(c)

図2.8　２の補数の作り方

図2.8で(b)の状態と(a)の状態を較べると，０と１が入れ換わっています．２進数の減算で１ビットだけであれば，

$$1-1=0$$
$$1-0=1$$

のように桁借りは生じません．つまり，２の補数を作るには，各桁の０と１を入れ換えてその後に１を足せばよいのです．

補数の作り方はこれくらいにしておいて，実際に計算を行ってみましょう．補数を使用した減算――つまりこの場合は加算となるわけですが――は最上位の桁上がりが生じたときはこれを無視します．また桁上がりが生じなかったときは，得られた結果は補数となります．先程２進数の0110（10進数の６）の補数を作りましたから，これを利用した例を挙げます(図2.9参照)．

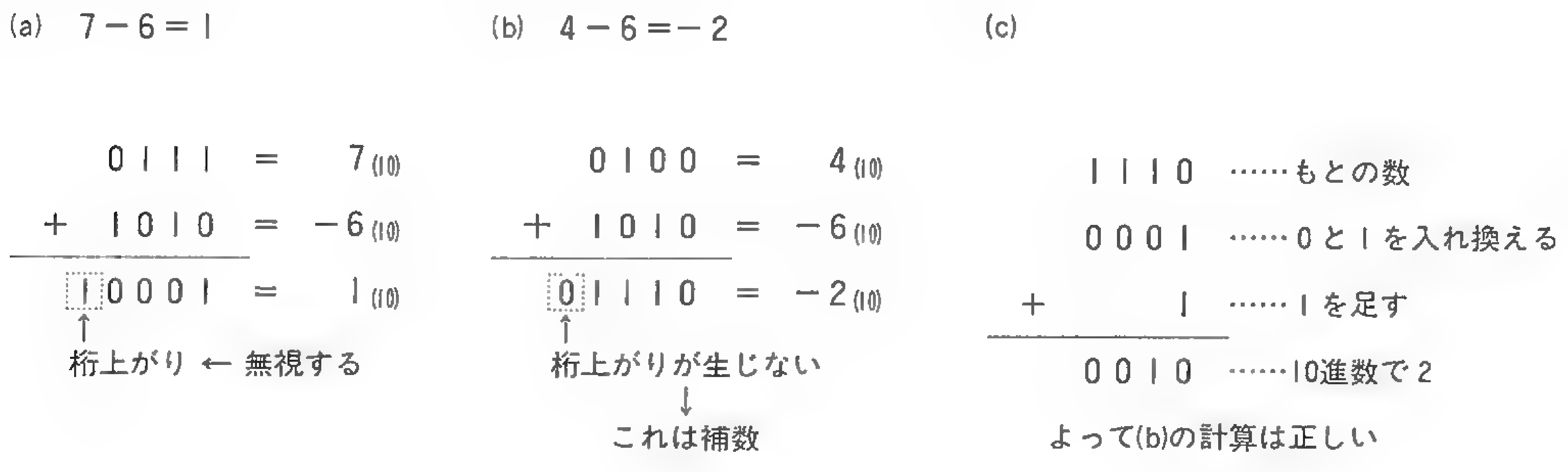

(a)　7－6＝1
0111　＝　7(10)
＋　1010　＝　－6(10)
10001　＝　1(10)
桁上がり ← 無視する
(b)　4－6＝－2
0100　＝　4(10)
＋　1010　＝　－6(10)
01110　＝　－2(10)
桁上がりが生じない
これは補数
(c)
1110 ……もとの数
0001 ……0と1を入れ換える
＋　1 ……1を足す
0010 ……10進数で2
よって(b)の計算は正しい

図2.9　補数の計算

(a)の例は，最上位に桁上がりが生じたので，これを無視します．すると答えは1となります．また，(b)の例は桁上がりが生じなかったので，得られた答えは補数表現となっており，この補数表現が－2であるか否かを確認しているのが(c)です．(c)のように，補数をもとに戻すのも補数を作るのと同じ方法で行えます．

このように，補数を使えば加算で減算と同じ結果が得られます．このため，補数表現はコンピュータにおいて数を表現する場合に広く用いられます．例えば4ビットを使用した2の補数表現は，図2.10のようになります．

10進数	2進数
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
－1	1111
－2	1110
－3	1101
－4	1100
－5	1011
－6	1010
－7	1001
－8	1000

図2.10　2の補数表現(4ビット)

ここで，MSBに注目してください．0～7は，MSBが0，－1～－8は，MSBが1となっています．つまり正の数はMSBが0，負の数はMSBが1となるのが2の補数の特徴です．逆に言えば，MSBが1か0かを調べれば正負が判断できます．

2.5　2進数の演算(2)

前項で，加算，減算および補数について説明しましたのでここでは，乗／除算と，それに伴うテクニックとしてシフト演算について説明しましょう．

2.5.1　乗算

まず乗算の例を挙げます．10進数の13×10＝130を計算してみましょう(図2.11参照)．

```
    1101   ………10進数で13
    1010   ………10進数で10
 --------
    1101
  1101
 --------
10000010   ………10進数で130
```

図2.11　２進数の乗算

非常に簡単です．何しろ九九（くく）に相当するもの〝一一〟（いちいち）（？）は１×０，１×１の二通りしかありません．

２．５．２　除算

次に除算の例として，まず余りのない場合を挙げます．10進数の90÷９＝10を計算してみましょう（図2.12参照）．

```
            1010
      ---------
1001) 1011010
      1001
      ---------
        1001
        1001
      ---------
           0
```

図2.12　２進数の除算（1）

答えの1010（10進数で10）が得られます．

今度は，余りのある例です．10進数で67÷７＝９･･･4を計算してみます（図2.13参照）．

```
         1001
      ---------
0111) 1000011
      0111
      ---------
         1011
         0111
      ---------
         0100
```

図2.13　２進数の除算（2）

答えは上記のように1001余り0100（10進数で９余り４）となります．

２．５．３　シフト演算

さて，一通り２進法の四則演算の説明が終わりましたので，少し風変りな計算方法を紹介しましょう．

２進数を２倍するには，どうすればよいのでしょう．もちろん a＋a＝2a となりますから，同じ数を加えればよいのです．試しに，２進数の１を２倍して，それを繰り返してみましょう．

```
1＋1＝10             (10進数で 2)
10＋10＝100          (   〃    4)
100＋100＝1000       (   〃    8)
1000＋1000＝10000    (   〃   16)
```

よくみると，０が１つずつ増えていくことがわかります．考えてみれば当然の話で，２進数を扱っているのですから，10進数の場合にある数を10倍するのに一番下の桁に０を付け加えればよいのと同じことです．ですから，２進数で２倍するには０を１つ，４倍にするには０を２つ，８倍するには０を３つ………というように０を付け加えていけばよいのです．

では，実際に確認してみます，図２.14をみてください．

```
     n   ・・・・・1101
    ...................

     2n       11010
 +) 8n      1101000
    ---------------
    10n    10000010
```

図2.14　左シフト

このように０を１個ずつ付け加える，言い換えれば１ビットずつ左へ移すための演算を左シフトといいます．シフト(Shift)には，移す，ずらすなどといった意味があり，シフト演算にはこの他に右シフトがあります．

右シフトは右へ１ビットずつ移動させるものですが左シフトと異なり，右へ１ビットずらせば値が必ずしも$\frac{1}{2}$，$\frac{1}{4}$となるわけではありませんので注意してください．これは実際の例をみれば明らかです．図2.15をみてください．(a)の場合は問題なく$\frac{1}{2}$，$\frac{1}{4}$になりますが，(b)の場合にはそうなっていません．

```
(a)                                  (b)
  1100                                 1101
  ------------                         ------------
   110|0        1回右シフト              110|1
    11|00       2回右シフト               11|01
```

図2.15　右シフト

このシフトという考え方は，2進数で乗／除算を行う場合にたいへん重要なテクニックとなります．ここでは演算の1つの手法として考えていただければ結構ですが，後程命令の説明で再度登場します．

2.6　BCDコードとは

BCDは，Binary Coded Decimal の略で，日本語に訳すと，2進化10進数，つまり2進数で表された10進数という意味です．具体的にはどのようなものかというと，2進数の4ビットを使用して，10進数の1桁を表すものです．表2.4を参照してください．10進数とBCDは，表のように対応付けられます．

10進数	BCD
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

表2.4　BCDコード表

この表をみると，2進数4ビットで表すことのできる残り6個のコード1010, 1011, 1100, 1101, 1110，1111は対応付けられていません．これらのコードは〝組み合わせ禁止(Forbidden Combination)〟と呼ばれ，BCDを扱う場合には使用してはいけないことになっています．図2.16に，BCDで表現した数の例を示します．

10進数	BCD		
2 5		0 0 1 0	0 1 0 1
7 3		0 1 1 1	0 0 1 1
1 2 3	0 0 0 1	0 0 1 0	0 0 1 1
7 8 0	0 1 1 1	1 0 0 0	0 0 0 0

図2.16　10進数とBCD

2.6.1　BCDの特徴

BCDは10進数との変換が容易で計算の誤差が生じないため，計算処理に利用されることが多いのですが，若干欠点があることは否めません．この欠点は実際にBCDを使用して計算してみればすぐ理解できます．

BCDの長所

(1)10進数との対応がわかりやすい

(2)10進数 ↔ BCD変換が簡単である

(3)10進数の演算が実行しやすい

BCDの短所

(1)コンピュータは２進演算を行うため演算が複雑になる

(2)計算速度が遅い

表2.5　BCDの特徴

2.6.2　BCDの計算

図2.17の(a)の例は普通に計算できるので，全く問題はありません．しかしながら，(b)の例は，答えが1100となり組み合わせ禁止となってしまいます．このような場合〝＋６補正〟というものが必要となります．これは組み合わせ禁止が６種類あるために，２進数４ビットで表せる数とのギャップを埋めるものです．先程の(b)の計算をもう一度見直してみます．図2.18をみてください．

```
(a)  6＋2＝8                        (b)  7＋5＝12

      0110 ……BCDで6                   0111 ……BCDで7
  ＋  0010 ……BCDで2               ＋  0101 ……BCDで5
  ─────────                       ─────────
      1000 ……BCDで8                   1100 ……BCDで？？
```

図2.17　BCD加算

```
7＋5＝12

        0111 ……BCDで7
   ＋   0101 ……BCDで5
   ──────────
        1100 ……BCDにおいて組み合わせ禁止
＋      0110 ……＋6補正
───────────────
    00010010 ……BCDで12
```

図2.18　BCD＋６補正

これで正しい答えが得られました．次に減算を行ってみます．

```
(a)  9－4＝5                       (b)  15－6＝9

      1 0 0 1  ……BCDで9             0 0 0 1 0 1 0 1  BCDで15
  －  0 1 0 0  ……BCDで4         －  0 0 0 0 0 1 1 0  BCDで6
  ───────────                   ─────────────────────
      0 1 0 1  ……BCDで5             0 0 0 0 1 1 1 1  BCDで？？
```

図2.19　BCD減算

(a)は別段支障なく減算が実行できた例ですが，(b)は，加算の際の(b)と同じく正常な計算が行えない例です．このような場合，加算の際に＋6補正が必要であったのと同様に，減算の場合には〝－6補正〟を行います．

```
        15－6＝9

        0 0 0 1 0 1 0 1  ……BCDで15
    －  0 0 0 0 0 1 1 0  ……BCDで6
  ─────────────────────
        0 0 0 0 1 1 1 1  ……BCDにおいて組み合わせ禁止
    －  0 0 0 0 0 1 1 0  ……－6補正
  ─────────────────────
        0 0 0 0 1 0 0 1  ……BCDで9
```

図2.20　BCD－6補正

このように，BCD演算を行うには必ず補正がつきまといますので，かなりめんどうです．当然のことながら，2進数で同じ演算を行うのに較べると，計算に時間がかかります．この負担を少しでも減らすために，マシン語の命令には，DAAというBCD演算の補正を自動的に行う命令が用意されていますが，それでもやはり実行速度が遅くなります．しかし，先程の長所を活かせば，10進数の計算はかなり実行しやすくなります．このようなことから，BCDは，電卓，事務のデータ処理などに広く利用されているようです．

2.7　論理演算

コンピュータで用いられる論理演算は，0と1だけから成る演算です．一般の算術演算には，加／減／乗／除算の4種類がありました．それに対して，論理演算には，基本的には，次の3種類があります．

(1)　論理和(OR)
(2)　論理積(AND)
(3)　否　定(NOT)

2.7.1　論理和(OR)

論理和の具体的な例として電球とスイッチの配線の例を挙げます．図2.21をみてください．A, B, ２つのスイッチが並列，スイッチと電球，電池が直列に接続してあります．ここで，スイッチＡ，Ｂは，開いているときが０，閉じているときが１，電球Ｙは，点灯しているときが１，消灯しているときが０と約束します．

(a)の場合をみてください．これは，スイッチＡ，Ｂ共に開いています．このとき，電球は当然消灯したままです．ですから，Ａ＝０，Ｂ＝０のときＹ＝０です．

(b)の場合は，スイッチＡのみが閉じています．Ｂは開いたままです．このときもちろん電球は点灯します．ですから，Ａ＝１，Ｂ＝０のとき，Ｙ＝１です．

(c)は，(b)とは逆にスイッチＡが開いてＢが閉じている場合です．このときも電球は点灯します．つまりＡ＝０，Ｂ＝１のときＹ＝１です．

最後に，(d)の場合ですがこれは，スイッチＡ，Ｂ共に閉じているときです．この場合も電球は点灯しますから，Ａ＝１，Ｂ＝１のときＹ＝１です．

OR回路

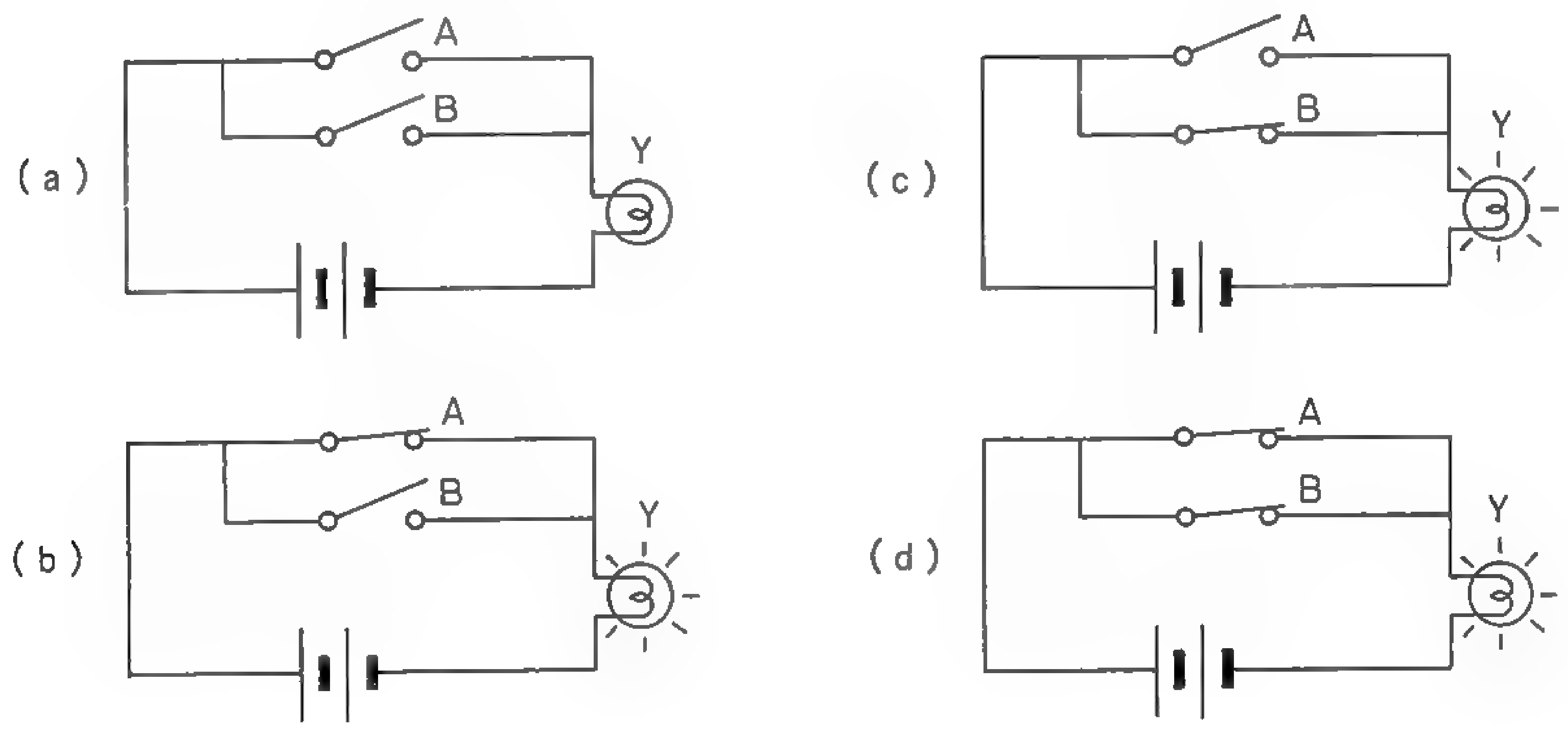


図2.21　OR

以上の結果を表にまとめてみます．

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

表2.6　ORの真理値表

表2.6は，真理値表(Truth Table)といいます．またA，B，Yに表２．６の関係があるときYは，AとBの論理和(OR)であるといいます．論理和の表現を式で書くと次のような表現法があります．

$$Y=A+B \qquad Y=A\vee B \qquad Y=A\cup B$$

Y＝A＋Bは，普通の加算とまぎらわしいので，注意してください．一般にはY＝A∨Bという表現が多く使われるようです．

2.7.2　論理積(AND)

論理和での例と同様の約束があるとして，図2.22をみてください．

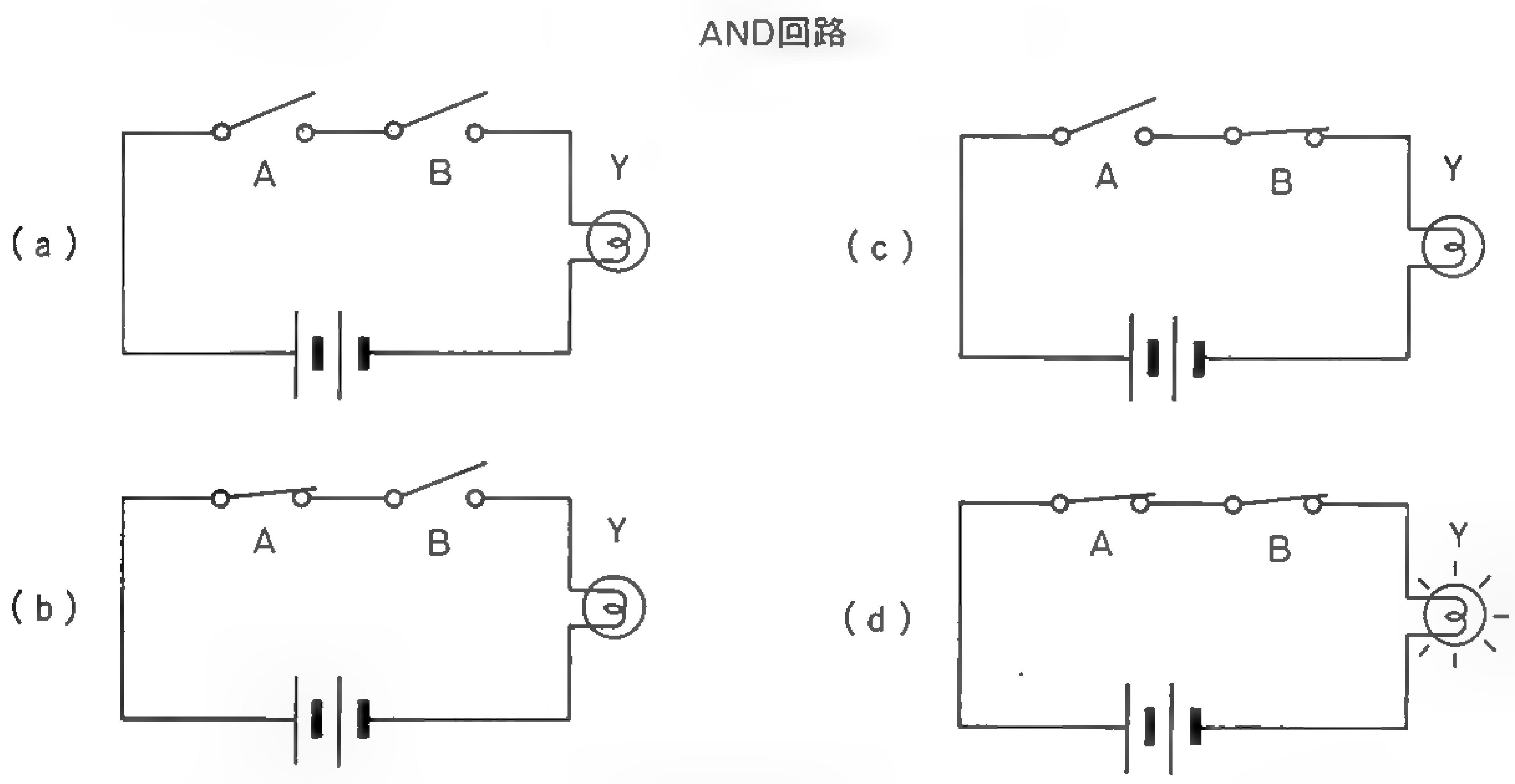


図2.22 AND

スイッチA，B，電球Y，電池がすべて直列に接続されています．(a)の場合スイッチA，B共に開いていますから，電球は消灯のままです．つまりA＝0，B＝0のとき，Y＝0です．

(b)の場合は，スイッチAは閉じていますが，Bが開いているので電球は消灯のままです．A＝1，B＝0のとき，Y＝0です．

(c)は(b)の逆の場合です．やはり電球は消灯のままですから，A＝0，B＝1のとき，Y＝0です．

(d)の場合，スイッチA，Bどちらも閉じていてこれでやっと回路がつながり，電球Yが点灯します．A＝1，B＝1のとき，Y＝1です．

A，B，Y間に表2.7の関係があるとき，YはAとBの論理積(AND)であるといいます．論理積の表現を式で書くと，次のような表現法があります．

$$Y=A\times B \qquad Y=A\wedge B \qquad Y=A\cap B$$

普通の乗法との混同を防ぐためにY＝A∧Bが比較的多く用いられます．

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

表2.7　ANDの真理値表

2.7.3　否定(NOT)

最後は，否定(NOT)です．否定とは，現在の状態を反転するものです．

1の否定は　0
0の否定は　1

否定の表現は一般に$\overline{A}$が用いられます．真理値表は，表2.8のようになります．

A	$\overline{A}$
0	1
1	0

表2.8　NOTの真理値表

2.7.4　論理演算の値

さて，BASICの演算子にもAND，OR，NOTがあります．ちょっとこれらの動作を確認してみましょう．

ダイレクトステートメントで次の式を実行してみてください．答えは1が得られると思います．

```
PRINT  1  AND  1 ↵
```

それでは次の例はどうなるでしょう．

```
PRINT  123  AND  123 ↵
```

これは，123と表示します．

次に，異なる数値の論理積をとります．

```
PRINT  123  AND  234 ↵
```

答えは，何と106になりました．これは，次のような理由によります．

$123_{(10)}$を２進数に変換すると　111 1011

$234_{(10)}$を２進数に変換すると　1110 1010

0 1 1 1 1 0 1 1 ……10進数で123

1 1 1 0 1 0 1 0 ……10進数で234

0 1 1 0 1 0 1 0 ……10進数で106

図2.23　ANDの実行例

図2.23をみてください．各ビットは，同じ桁にあたるものの論理積がとられていることがわかります．このように，BASICでは２進数で表された同じ桁どうしの論理積がとられます．これは，論理和，否定についても同様です．ただし，BASICの論理演算で用いられる２進数は，16ビットを単位としているので，扱うことのできる数は，−32768から32767までです．

以上の３つの例でおわかりいただけたと思いますが，論理演算と２進数にはこのような深い関係があります．

2.7.5　排他的論理和

ところで，論理演算の基本的な３種類の組み合わせによって作られる演算があります．これは，排他的論理和（eXclusive OR 略して XOR）と呼ばれる演算です．排他的論理和の真理値表を表2.9に示します．

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

表2.9　XORの真理値表

論理和の真理値表と較べると，Ａ＝１，Ｂ＝１のときＹ＝０となるところが異なっています．排他的論理和の定義は，〝Ａの否定〞と〝Ｂ〞の論理積 をとったものと，〝Ａ〞と〝Ｂの否定〞の論理積の論理和をとったものです．

$$Y = \overline{A} \land B \lor A \land \overline{B}$$

この場合の演算順序は論理和より論理積の方が優先されます．排他的論理和の演算子は$\forall$か$\oplus$で表されます．

$$Y = A \forall B \quad Y = A \oplus B$$

A	B	$\overline{A}$	$\overline{B}$	$A\wedge B$	$A\vee B$	$\overline{A}\wedge B$	$A\wedge\overline{B}$	$(\overline{A}\wedge B)\vee(A\wedge\overline{B})=A\forall B$
0	0	1	1	0	0	0	0	0
0	1	1	0	0	1	1	0	1
1	0	0	1	0	1	0	1	1
1	1	0	0	1	1	0	0	0

表2.10　真理値表のまとめ

2.7.6　論理演算の利用法

さて，論理演算は，何に利用されるのでしょうか．算術演算は確かに利用法がありそうです．しかし，AND，OR，NOT，XORは，ちょっとみると利用価値が全くないような気がします．ところが，マシン語でプログラムを組むときは，これらの論理演算はいろいろと利用されるのです．

ある2進数のデータで，図2.24の(a)のようなデータがあったとします．このデータの下位4ビットすべてを1にしたい場合，これは(b)のデータを用意して加えれば，目的のデータが得られます．しかし，これでは(a)のデータの内容が変わったとき(b)のデータも変えなければならず，プログラミングがたいへんです．そこでこういった場合には，上位4ビットが0，下位4ビットが1のデータ(c)を用意します．そして，(a)と(c)のORをとると，目的のデータが得られます．つまり(a)のデータ内容がどのように変わろうとも，下位4ビットは常に1になります．どうしてこのようになるかは，ORの真理値表を思い出してください．Aが1のとき，Bが0であっても1であっても，Yは1となります．また，Aが0のときはYの値はそのままBの値となります．

(a)	0 1 0 1 0 1 0 1
(b)	0 0 0 0 1 0 1 0
(c)	0 0 0 0 1 1 1 1
(a)+(b)	0 1 0 1 1 1 1 1
(a)$\vee$(c)	0 1 0 1 1 1 1 1

図2.24　ORの利用

ORの真理値表

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

このように，ORはあるデータの特定のビットを1にしたいときに用いられます．

それでは，ANDはどうでしょう．ANDの真理値表を思い出してください．Aが0のときは，Bの値が0，1のどちらであってもYは0となります．また，Aが1のときYの値はそのままBの値となり，ORとは逆に特定のビットを0にするのに用いられます．具体例を図2.25に示します．

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

(a) 1 1 0 1 0 1 1 0　　(b) 0 1 1 1 0 0 1 0

上位４ビットを０にするには

(c) 0 0 0 0 1 1 1 1

(a)∧(c) 0 0 0 0 0 1 1 0　　(b)∧(c) 0 0 0 0 0 0 1 0

図2.25 ANDの利用

ORが任意のビットを1，ANDが0にするのに対し，XORは，反転(NOT)するのに用いられます．すべてのビットを反転するにはNOTを使用すればよいのですが，任意のビットだけを反転する場合，NOTでは不可能です．XORの真理値表をみると，Aが0のとき，Yの値はBの値と等しくなりますが，Aが1のときは，Bが1ならYは0，Bが0ならYは1と，Yの値はBの値のNOTをとったものとなります．この例を図2.26に示します．

(a) 0 1 1 0 1 1 0 0

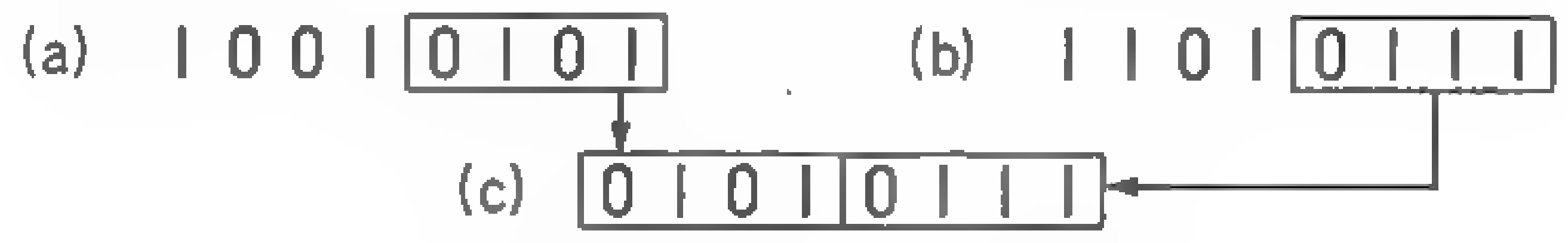
下位４ビットを反転する

(b) 0 0 0 0 1 1 1 1

(a)⊻(b) 0 1 1 0 0 0 1 1

図2.26 ビットの反転

最後に，AND，ORおよびシフトの使用例を図2.27に示します．この例では，データ(a)の下位４ビットとデータ(b)の下位４ビットを合成して，データ(c)を作ります．



(a)∧0 0 0 0 1 1 1 1 = 0 0 0 0 0 1 0 1 ……(d)

(d)を４ビット左にシフト 0 1 0 1 0 0 0 0 ……(e)

(b)∧0 0 0 0 1 1 1 1 = 0 0 0 0 0 1 1 1 ……(f)

(f)∨(e) = 0 1 0 1 0 1 1 1 =(c)となる

図2.27 論理演算の利用法

2.8 プログラミングのためのハードウェア知識

コンピュータといえば電子機器の代表製品のようなものですから，素人にとってそのハードウェアの知識をすべて知ることはとうてい不可能なことです．また，本書ではシステム・プログラム(キーボードやスクリーンなどを管理する最も基本的なプログラムなど)を作成することを目的

としているわけではありませんので，ハードウェアについての必要な知識も微々たるものです．

そこでここでは，最低限必要な知識としてCPU，メモリ，レジスタ，アドレス，Ｉ／Ｏ，Ｉ／Ｏポートについて説明します．

２．８．１　CPU

CPUとは，メモリに書き込まれた一連の命令やデータ（プログラム）をメモリから順次読み出して解読を行い，プログラムに従った動作を行うものです．そのためCPUにはいままで説明してきたような２進数の演算や論理演算を処理する機能が備わっており，CPU内のレジスタに一時的に記憶された内容の演算処理を行い，処理結果を各種装置（I／Oポートやメモリ）とやりとりしているのです．

２．８．２　メモリ（記憶装置）

メモリとはコンピュータに実行させるプログラム，つまり命令やデータの記憶を行います．CPUは，メモリに記憶されたプログラムにより，これを忠実に実行します．

メモリの種類には大きく分けてRAM（ラム）とROM（ロム）の２種類があります．

▶RAM

RAMは，Random Access Memoryの略で，読み書き可能なメモリのことです．日本語では，等速呼出メモリといいますが，この意味においては，ROMはRAMの一種ということになりますから，ROMと区別するためにはRWM（Read Write Memory）と呼ぶこともあります．しかし，一般にRAMという場合，RWMのことを指します．

RAMは，ごく一部の種類のものを除くと，電源を切ると記憶内容が破壊されてしまうという欠点があります．つまり，記憶内容を保持するためには，電源を接続したままにしなければならず，たいへん不便です．

▶ROM

ROMとは，Read Only Memoryの略で，読み出し専用メモリということになります．つまり，読むことはできても書くことはできません．するといちばん最初の内容はどうやって書き込むのか，という疑問が生じますが，これにはROMの種類によって２つの方法があります．

１つはマスクROMの場合です．これは，工場出荷の時点ですでに内容が書き込まれており，ユーザが書き換えることはできません．内容は，メーカに直接依頼して書いてもらうことになりますから，個数が大量でないと，メーカが引き受けてくれません．これに対して，P-ROM（ピーロム）というものがあります．P-ROMのＰは，Programableの略です．これは，市販されている時点では，内容は何も書き込まれておらず，ユーザが自分で書き込めるようになっています．このための道具が，P-ROMライタと呼ばれるものです．大部分のP-POMは，書き込む際に，

普通の電源（＋5Vか＋12V）より高い電圧(＋20V以上)を必要とします．マスクROMは，内容を消したり書き換えたりできませんが，P-ROMは紫外線を当てたり，高電圧をかけたりすれば，内容を消して再び別の内容を書くことのできるものもあります．またP-ROMの中には，時間がたつと内容の消えてしまうものがありますが，普通に使用する分には充分長い時間(10年以上)保持できますので，まず困ることはないでしょう．

2.8.3　アドレス(番地)

メモリは，非常に多くの記憶場所をもっているために，それぞれ2バイト(16進数で4桁)の数字に対応させて区別しています．これは，1.1で表示した画面の左端に出てきた4桁の数字のことです．

16進数4桁では，16^4＝65536の区別ができるのですから，アドレスは65536個のメモリに割り当てることができます．つまり，16進数では0000H～FFFFHの数字で表現されます．

2.8.4　I／O(入／出力装置)

I／O(アイオー)は，Input／Outputの略で，人間にたとえると，手，足，目，耳などにあたるもので，人間と，マイクロコンピュータとの橋渡しをする部分です．つまり，マイクロ・コンピュータに何かを入力したり，コンピュータの実行した結果を表示するものをいいます．これらのうち，I／OのI，つまり入力装置に相当するものには，キーボード，ライトペン，ジョイスティック，トラックボール，デジタイザ等があります．また，I／OのOつまり出力装置に相当するものには，スクリーン，プリンタ，XYプロッタ，スピーカ，ブザー等があります．

2.8.5　I／Oポート

CPUの内部と外部との仲介役を果たし，CPUで処理した結果を外部に出力したり，外部からのデータを内部へ読み込むときのデータの出入口です．I／Oポートにはメモリと同様に00H～FFHまでのアドレスが付けられており，最大256個のI／O装置をコントロールすることができます．私たちがI／Oを利用する場合には，マシン語の命令やモニタのコマンドを利用してI／Oポートを操作します．

2.8.6　レジスタ

レジスタについては第3章のCPUとマシン語のところで詳しく述べますが，簡単にいえばCPUがプログラムを実行する上で必要となる様々なデータの一時的な記憶場所として使用されるものです．BASICではプログラム中でデータを扱う場合に変数を用いますが，レジスタはマシン語で使う変数と考えていただければ結構です．

2.8.7　情報の流れ

さて，コンピュータの主要な部分の説明を行いましたが，コンピュータとして機能するためには，これらの各部分が単独で機能するのではなく，情報(プログラム，データ)の流れに従って働かなければなりません．一般的なコンピュータを例にした場合，概念的には図2.28のような構成になり，実線で結ばれた部分が情報のやりとりを示しています．

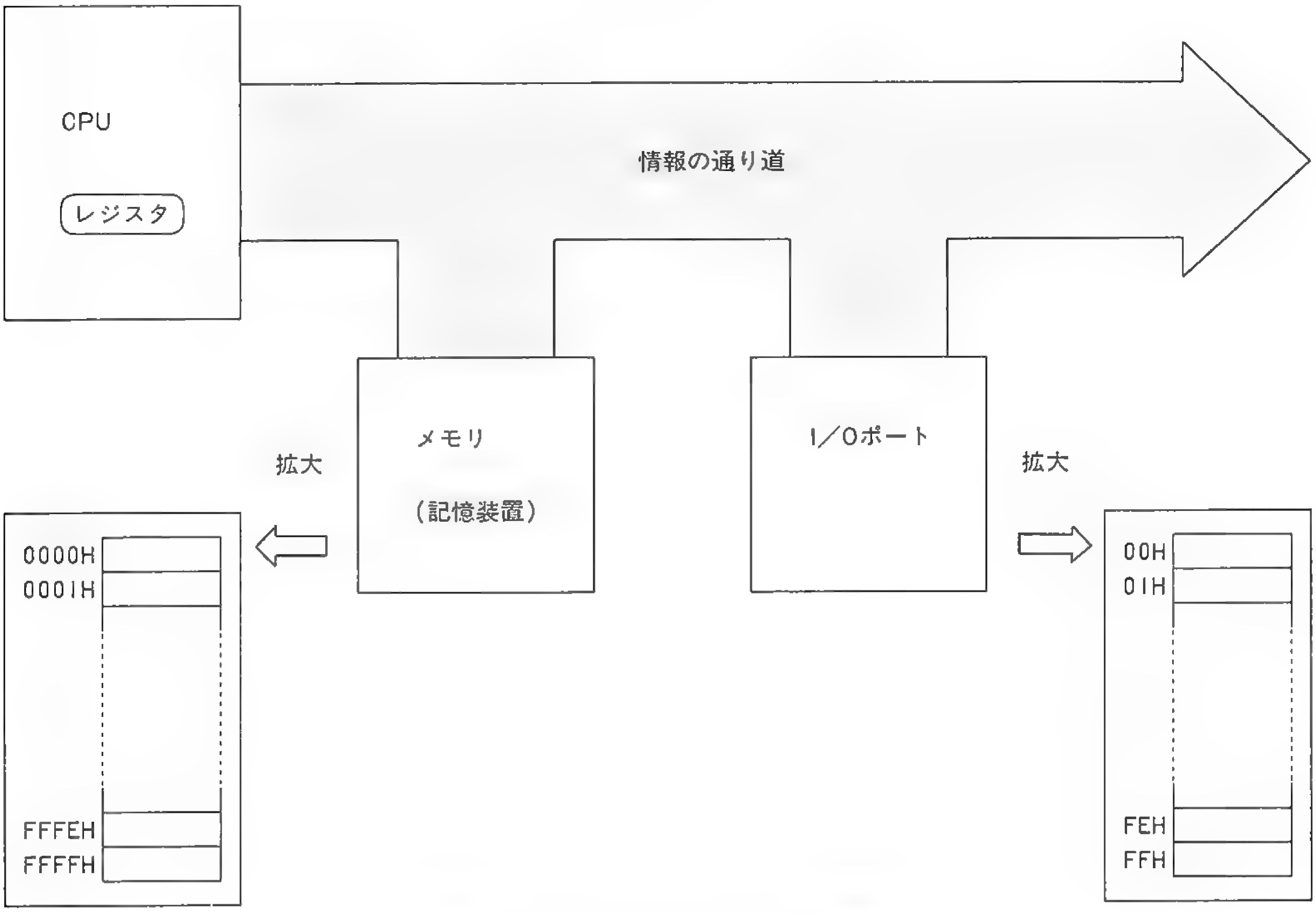


図2.28　マイクロコンピュータの構成

2.9　符号による表現

本章の初めにBCDコードについて説明しました．これは，10進数を扱いやすくするための１つの手法として考えられたものです．これと同様に，数字以外の文字，つまりアルファベットや，"？"，"！" などの記号(これはアメリカでの話ですからもちろんカナなどは入っていません）をコードで表そうとして考えられたものが，アスキー・コードです．

ASCIIとは，American Standard Code for Information Interchangeの略で，日本語に訳すと，情報交換のためのアメリカ標準コードという意味になります．アスキー・コード自体は，７ビットなのですが，大抵の場合，パリティビットを最上位に付けて８ビットで用いられます．パリティビット(Parity Bit)は，コード内の１と０の数と，パリティビットまで含めて数えて，１の数を偶数にしたものが偶数パリティといい，１の数を奇数にしたものが奇数パリティといいます．パリティについては，次章のフラグのところでも説明しますのでそちらを参照してください．

アスキー・コードを基本にして，カナを加えてJISが定めたのが，日本で一般的に使われているJISコードです．現在市販されているパソコンの大部分はこのJISコードを基にキャラクタ・コードを作成しています．

下位＼上位	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DEL	SP	0	@	P	／	p				ー	タ	ミ		
1	SOH	DC1	!	1	A	Q	a	q			。	ア	チ	ム		
2	STX	DC2	″	2	B	R	b	r			「	イ	ツ	メ		
3	ETX	DC3	#	3	C	S	c	s			」	ウ	テ	モ		
4	EOT	DC4	$	4	D	T	d	t			、	エ	ト	ヤ		
5	ENO	NAK	%	5	E	U	e	u			・	オ	ナ	ユ		
6	ACK	SYN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7	BEL	ETB	'	7	G	W	g	w			ァ	キ	ヌ	ラ		
8	BS	CAN	(	8	H	X	h	x			ィ	ク	ネ	リ		
9	HT	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A	LF	SUB	*	:	J	Z	j	z			ェ	コ	ハ	レ		
B	VT	ESC	+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C	FF	FS	,	<	L	¥	l	|			ャ	シ	フ	ワ		
D	CR	GS	−	=	M	]	m	}			ュ	ス	ヘ	ン		
E	SO	RS	.	>	N	^	n	—			ョ	セ	ホ	゛		
F	SI	US	／	?	O	—	o	DEL			ッ	ソ	マ	゜		

表2.11　JISコード表

下位＼上位	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		D_E		0	@	P		p	▁	┴		ー	タ	ミ	═	×
1	S_H	D_1	!	1	A	Q	a	q	▂	┬	。	ア	チ	ム	╞	円
2	S_X	D_2	″	2	B	R	b	r	▃	┤	「	イ	ツ	メ	╪	年
3	E_X	D_3	#	3	C	S	c	s	▄	├	」	ウ	テ	モ	╡	月
4	E_T	D_4	$	4	D	T	d	t	▅	▔	、	エ	ト	ヤ	◢	日
5	E_Q	N_K	%	5	E	U	e	u	▆	─	・	オ	ナ	ユ	◣	時
6	A_K	S_N	&	6	F	V	f	v	▇	│	ヲ	カ	ニ	ヨ	◥	分
7	B_L	E_B	'	7	G	W	g	w	█	▕	ァ	キ	ヌ	ラ	◤	秒
8	B_S	C_N	(	8	H	X	h	x	▏	┌	ィ	ク	ネ	リ	♠	
9	H_T	E_M	)	9	I	Y	i	y	▎	┐	ゥ	ケ	ノ	ル	♥	
A	L_F	S_B	*	:	J	Z	j	z	▍	└	ェ	コ	ハ	レ	♦	
B	H_M	E_C	+	;	K	[	k	{	▌	┘	ォ	サ	ヒ	ロ	♣	
C	C_L	→	,	<	L	¥	l	:	▋	╭	ャ	シ	フ	ワ	●	
D	C_R	←	−	=	M	]	m	}	▊	╮	ュ	ス	ヘ	ン	○	
E	S_O	↑	.	>	N	^	n	~	▉	╰	ョ	セ	ホ	゛	╱	
F	S_I	↓	／	?	O	—	o		┼	╯	ッ	ソ	マ	゜	╲	

表2.12　PC-8801キャラクタ・コード表

バイトのはなし

PC-8801には，標準でRAM 112Kバイト，ROM 72Kバイトが実装されています．いつも何となくみる表現ですが，Kバイトという単位はいくつを表すのかご存知ですか？

一般にK(キロまたはケー)という単位は，基になる量の1000倍を表します．しかし，コンピュータを扱う場合のKは$2^{10}=1024$を表していますので，１Kバイトというのは1000バイトではなく，1024バイトというわけです．

同様にして，M（メガ）は，$(10^3)^2$ではなくて（$2^{10})^2$となるので，１Mバイトは1024Kバイト＝1,048,576バイトとなります．

さて，１Kバイトを，16進数にした場合はいくつになるのでしょうか．それを実際に調べてみましょう．

１Kバイト＝1024バイト＝2^{10}バイトですから16進数で400Hとなります．

$$1024_{(10)} = 100\quad 0000\quad 0000_{(2)}$$
$$1024_{(10)} = \quad 4\qquad 0\qquad 0_{(16)}$$

このあたりの量の表現は，メモリ計算等でよく利用するのでよく覚えておいてください．

下位＼上位	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P		p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	¦
C	FF	FS	,	<	L	／	l	[
D	CR	GS	−	=	M	]	m	¦
E	SO	RS	.	>	N	^	n	~
F	SI	US	／	?	O	_	o	DEL

表2.13　アスキー・コード表

第3章　マシン語基礎知識

第2章のマイコン基礎知識に引き続き，本章ではマシン語の基礎的な知識について説明します．

マシン語でプログラムを記述するには，BASICプログラミングでは必要のなかったことがいくつか登場します．その中には単に用語の説明で済むものもありますが，根本的にプログラミングの考え方が異なる点もあります．本章では実際にマシン語に関わる上で必要な点を一通り説明していますので，マシン語の学習を行うのに役立つと思います．

読者の皆さんは，以後の章で不明な点が出てきたら再度本章を読み返して，確認しながら学習されるとよいでしょう．

3.1　CPUとマシン語

第1章の初めで，「マシン語とは，CPUが直接実行することのできる唯一の言語だ」と述べました．マシン語は，一部の例外を除いて特定のCPUでしか理解できないので，μPD780の命令を覚えただけではFM-7の6809CPUやアップルIIの6502CPUのマシン語プログラムは作ることができません．ただし，あるCPUのマシン語プログラムについてよく理解していれば，他のCPUのマシン語を理解するのも比較的容易です．

3.1.1　CPUのいろいろ

現在市販されている8ビットのCPUの主なものに，インテル社の8080A，8085A，モトローラ社の6800，6802，6809，ザイログ社のZ80，モステクノロジー社の6502，およびそれらのセカンドソース(本来の製造メーカ以外の供給元)等があります．これらのうち，同一の会社で作られたCPUのマシン語に関しては若干の互換性がありますが，他社のCPUとは全くといってよい程互換性はありません．ただザイログ社のZ80に関しては，インテル社の8080Aを基本にして作られたために，特定機種のハードウェアに依存しない限り，8080Aのマシン語プログラムは，そのままZ80で走らせることができます．最初に述べたTK-80というワンボード・マイコンは，CPUに8080Aを使用していたので，CPUの機能をグレードアップするために，一時期Z80と交換するのがは

やった程です．Z80では，8080Aの命令に新たな命令を追加するという形で，マシン語の命令ができ上がっていますから，8080Aの命令よりもその命令体系が少々複雑で，命令数も8080Aの78に対し，Z80では158と多くなっています．

PC-8801に使用されているCPUは，Z80コンパチブルの高速タイプに相当するものです．高速タイプというのは，供給できるクロックの上限周波数が高くなっているものです．例えば，Z80に2.5MHzのクロックを供給すると，マシン語のNOPという命令を1.6μs(1.6×10^{-6}秒)で実行します．ところがクロックが4MHzであるZ80Aを使用すれば，この場合前述の命令は1μsで実行されます．このように，クロック周波数が高ければ高い程CPUの実行速度は向上しますが，上限周波数には限界があり，どんなに高い周波数でも動作するということはありません．また，CPUの種類によっても動作が全く異なりますので，実行速度の比較を簡単に行うことはできません．ハードウェアレベルでみた場合，FM-7のMBL68B09のクロック周波数2MHz(説明書には8MHzと書かれているが，MBL68B09は内部でクロックを4分周して，実質は2MHzとなる)は，Z80Aの4MHzより高速であるといえます．

3.2　μPD780のレジスタ構成

マシン語でプログラムを組む場合，非常に重要な役割を果たすのがCPUのレジスタ群の構成です．これをレジスタ構成と呼びます．レジスタは，前にも述べましたがデータを一時的に記憶するところであり，BASICにたとえると変数に相当するものです．

PC-8801に搭載されているCPUはμPD780と呼ばれるCPUですが，これはザイログ社のZ80Aと完全に互換性があります．つまり，レジスタ構成および命令構成は同一のものですから，Z80Aおよび8080Aで作られたマシン語プログラムは，特定のハードウェアに依存しない限りμPD780で利用できます．

μPD780のレジスタ構成(Z80のレジスタ構成はこれと全く同じ)は**表3.1**に，またその土台となった8080Aのレジスタ構成は，**表3.2**に示した通りです．

表3.1と**表3.2**を比較すると，A～FおよびH，Lレジスタ，プログラム･カウンタ(PC)，スタック・ポインタ(SP)は共通にありますが，μPD780には，A～FおよびH，Lレジスタは2組ずつありますし，その他にもインデックス・レジスタ(IX，IY)や，IおよびRレジスタがあります．また，Fレジスタの内容(フラグの構成)も少々異なっています．このようにμPD780は，8080Aのレジスタに他のレジスタを付け加えるといった構成をしており，ここに8080Aのマシン語プログラムがほぼそのまま走るという理由があります．つまり，8080Aには備わっていないレジスタを無視すれば，μPD780は，8080Aと機能的にほぼ同等といえます．ここでは各レジスタの機能について説明していきますが，上手なマシン語プログラムを作るためには，各々のレジスタ機能をよく理解して，今このレジスタにはどのようなデータを記憶して，何を行っているかを把握することが大切です．

レジスタ名	ビット	レジスタ名	ビット
Aレジスタ	8	Fレジスタ (フラグ・レジスタ)	8
Bレジスタ	8	Cレジスタ	8
Dレジスタ	8	Eレジスタ	8
Hレジスタ	8	Lレジスタ	8
プログラム・カウンタ	16	スタック・ポインタ	16
IXレジスタ	16	IYレジスタ	16
Iレジスタ	8	Rレジスタ	7

主レジスタ

レジスタ名	ビット	レジスタ名	ビット
A'レジスタ	8	F'レジスタ	8
B'レジスタ	8	C'レジスタ	8
D'レジスタ	8	E'レジスタ	8
H'レジスタ	8	L'レジスタ	8

副レジスタ

Fレジスタの構成

S	Z	×	H	×	P/V	N	CY

表3.1　μPD780CPU レジスタ構成

レジスタ名	ビット	レジスタ名	ビット
Aレジスタ	8	Fレジスタ	8
Bレジスタ	8	Cレジスタ	8
Dレジスタ	8	Eレジスタ	8
Hレジスタ	8	Lレジスタ	8
プログラム・カウンタ	16	スタック・ポインタ	16

Fレジスタの構成

S	Z	×	×	×	P	×	CY

表3.2　8080A CPU レジスタ構成

ＩレジスタとＲレジスタは，ほとんど利用されることがないので，ここでは説明を省略します．また，Ｆレジスタは次節で詳しく説明します．

3.2.1　主レジスタと副レジスタ

表3.1をみると，Ａ～Ｆ，Ｈ，Ｌレジスタは２組ずつあり，一方を主レジスタ，他方を副レジスタと呼びますが，主，副といっても特に機能が違うわけではなく，プログラマが同時に使えるのが主レジスタ側のみであるためこう呼ばれるのであって，交換命令によって，主レジスタと副レジスタを入れ換えることができます．そのため，主レジスタを表レジスタ，副レジスタを裏レジスタとも呼びます．

3.2.2　アキュムレータ

ソフトウェアからみると，大部分の演算がAレジスタで行われます.「ソフトウェアからみると」というのは，実際に演算を行うのはALUと呼ばれるものが行うのであって，Aレジスタにはその結果が保存されるために，みかけ上Aレジスタで演算が行われたようにみえるということです.

このように，Aレジスタはレジスタの中でも，演算を担う重要な位置をしめており，プログラムを組む場合にはその存在を無視するわけにはいきません．Aレジスタのように主に演算を目的に使用されるレジスタをアキュムレータ(Accumulater)といいます.

3.2.3　汎用レジスタ

B，C，D，E，HおよびLレジスタは，汎用レジスタと呼ばれ，単独では8ビットのレジスタですが，BC，DE，HLのそれぞれを対にして16ビットのレジスタ・ペアとして用いられることもあります．これらのレジスタは，ループカウンタとして用いられたり，HLレジスタ・ペアと，他のレジスタ・ペアの加／減算などが行えます.

3.2.4　インデックス・レジスタ

インデックス・レジスタは2つあり(IX，IY)，どちらも全く同じ機能をもった16ビットのレジスタです．これらのレジスタは，HLに代表されるレジスタ・ペアがその内容の指し示すアドレスのメモリしか扱うことができなかったのに対し，インデックス・レジスタの指し示すアドレスと命令に付加されるデータ(このデータをディスプレイスメント(displacement)と呼ぶ，ディスプレイスメントは8ビットであり，2の補数表示を用いる)を加えることにより，インデックス・レジスタの値を基準として＋127～－128のメモリのアドレスの内容を扱うことができます.

3.2.5　スタック・ポインタ

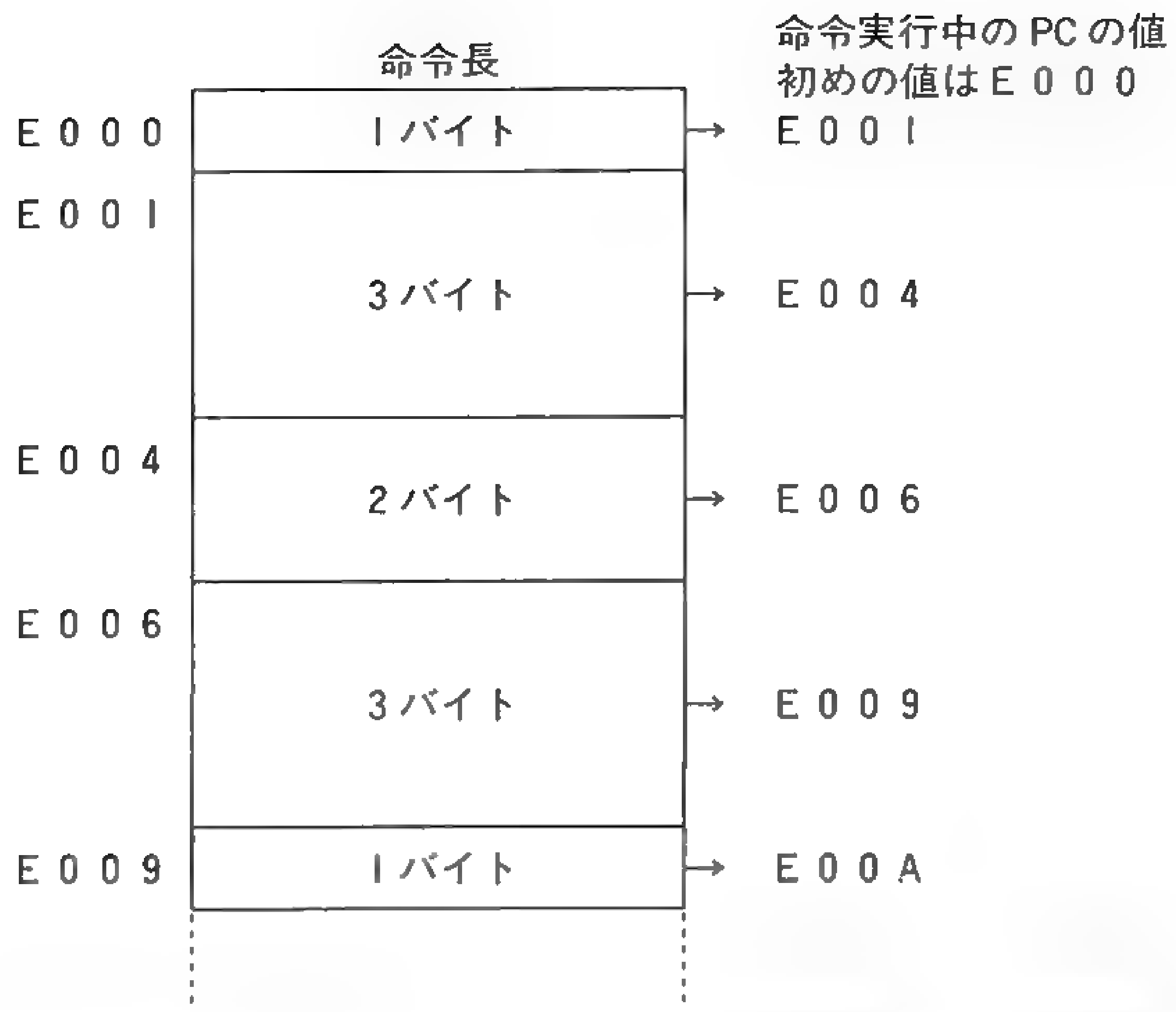
スタック・ポインタ(Stack Pointer：SP)は，メモリ上に設定されたスタックという特別な領域のアドレスを指し示すポインタです.

スタックとは，データやプログラム・カウンタの一時退避に使用され，最後に退避(PUSH)したデータを最初に取り出す(POP)ような形態をとっており，スタック・ポインタは，常にこのスタックトップのアドレスを指し示し，このような領域をLIFO(Last In First Out)といいます(第5章，PUSHの項参照).

3.2.6　プログラム・カウンタ

プログラム・カウンタ(Program Counter：PC)は，常に現在実行中の命令の次に実行する命令が格納されているアドレスを指し示します(図3.1参照).

前述の汎用レジスタに対し，IX, IY, SP, PC, I, Rをまとめて専用レジスタと呼びます.


命令長
命令実行中のPCの値
初めの値はＥ０００
Ｅ０００
1バイト
Ｅ００1
Ｅ００1
3バイト
Ｅ００４
Ｅ００４
2バイト
Ｅ００６
Ｅ００６
3バイト
Ｅ００９
Ｅ００９
1バイト
Ｅ００Ａ

数値はすべて16進数

図3.1　プログラム・カウンタの値

3.3　Ｆレジスタの役割

Ｆレジスタは，別名フラグ・レジスタとか，コンディション・コード(Condition Code)・レジスタと呼ばれる８ビットのレジスタです．Ｆレジスタは，他のレジスタがレジスタ自体で役割をもっていたのに対し，ビット単位で役割をもっているのが特徴です．図3.2にそのビット構成を示します．

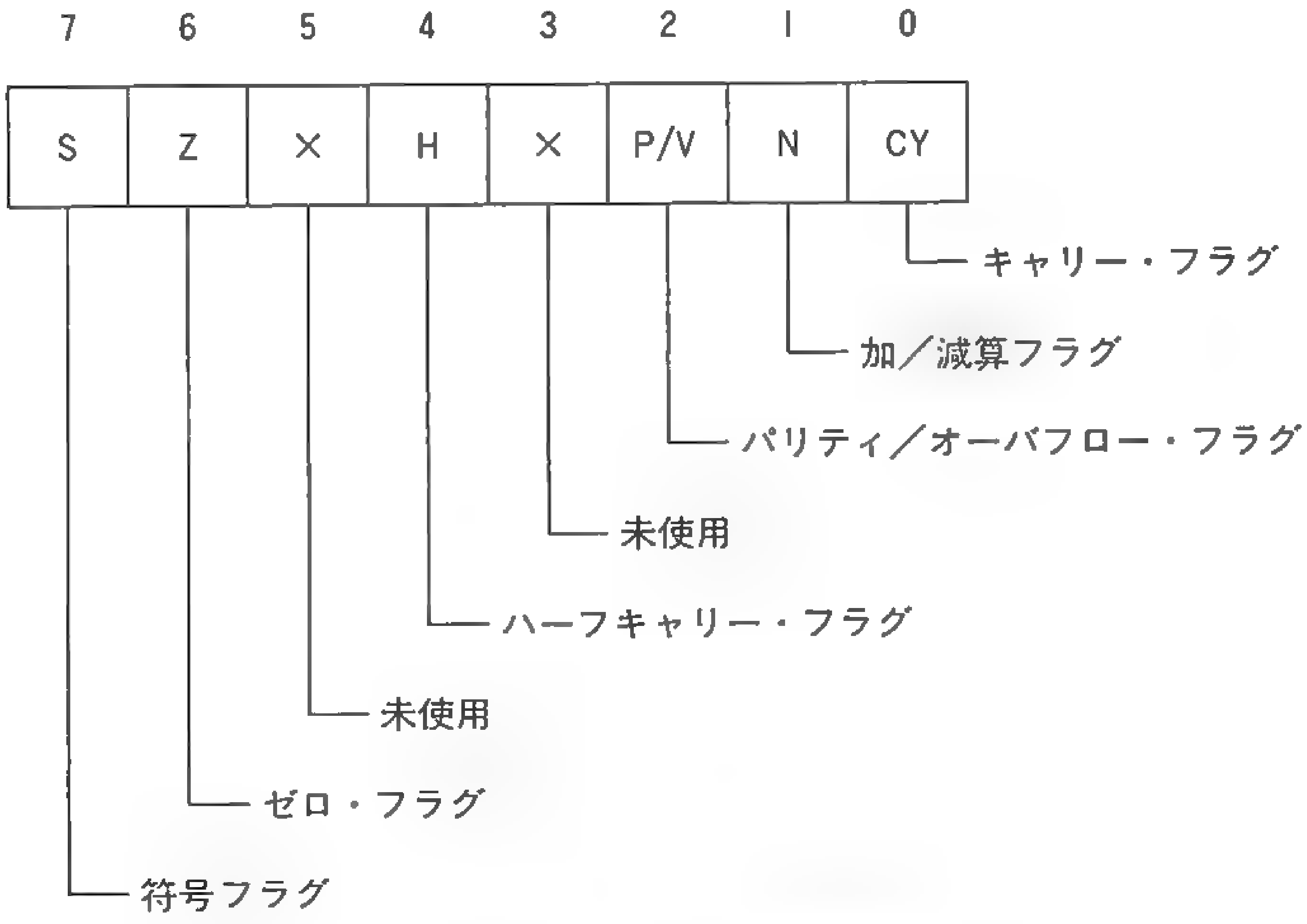

7
6
5
4
3
2
1
0
S
Z
×
H
×
P/V
N
CY
キャリー・フラグ
加／減算フラグ
パリティ／オーバフロー・フラグ
未使用
ハーフキャリー・フラグ
未使用
ゼロ・フラグ
符号フラグ

図3.2　Ｆレジスタの構成

Ｆレジスタの８ビットのうちビット３とビット５は未使用で，このビットの内容は不定です．つまり０なのか１なのかはっきりしないことになっています．その他のビットは，演算結果によって各ビットごとに変化します．このように，演算結果を反映させるものをフラグ，もしくはコ

ンディション・コードといい，これがフラグ・レジスタ，コンディション・コード・レジスタと呼ばれる理由です．

それでは各フラグの役割を説明しましょう．

3.3.1 CYフラグ

CYフラグは，キャリー・フラグ，または桁上がりフラグと呼ばれます．このフラグは，演算の結果最上位ビット(MSB)から桁上がりがあると1になります．そうでない場合は0です．また減算の場合，引かれる数が引く数より小さい場合に桁借りが生じ，CYフラグは1となります．

フラグに限らず，このようにビットの状態が1になることをそのビット(フラグ)がセットされるといい，その逆に0になるのを，ビット(フラグ)がリセットされるといいます．

CYフラグの動作の具体例を，図3.3に示します．

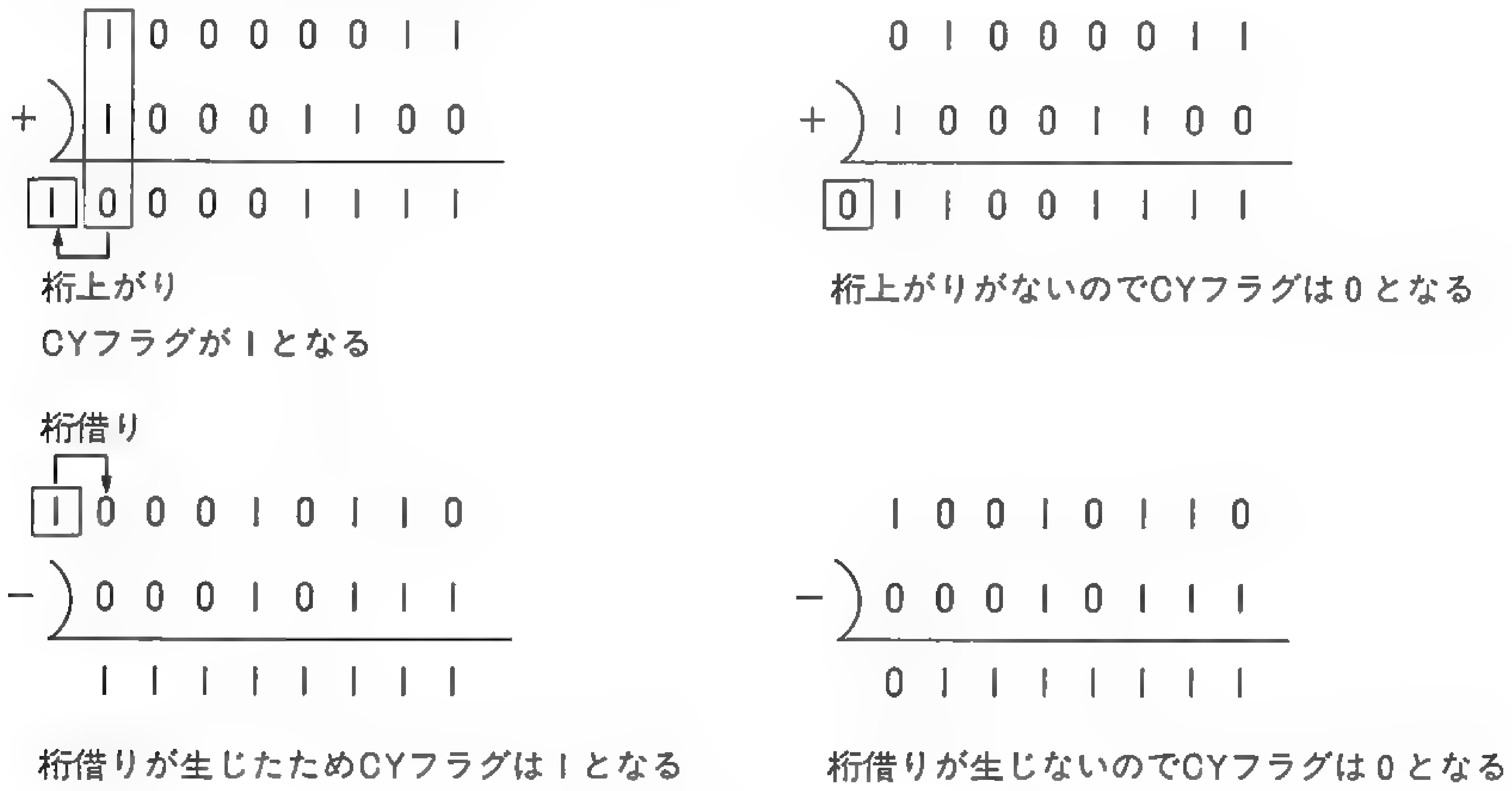
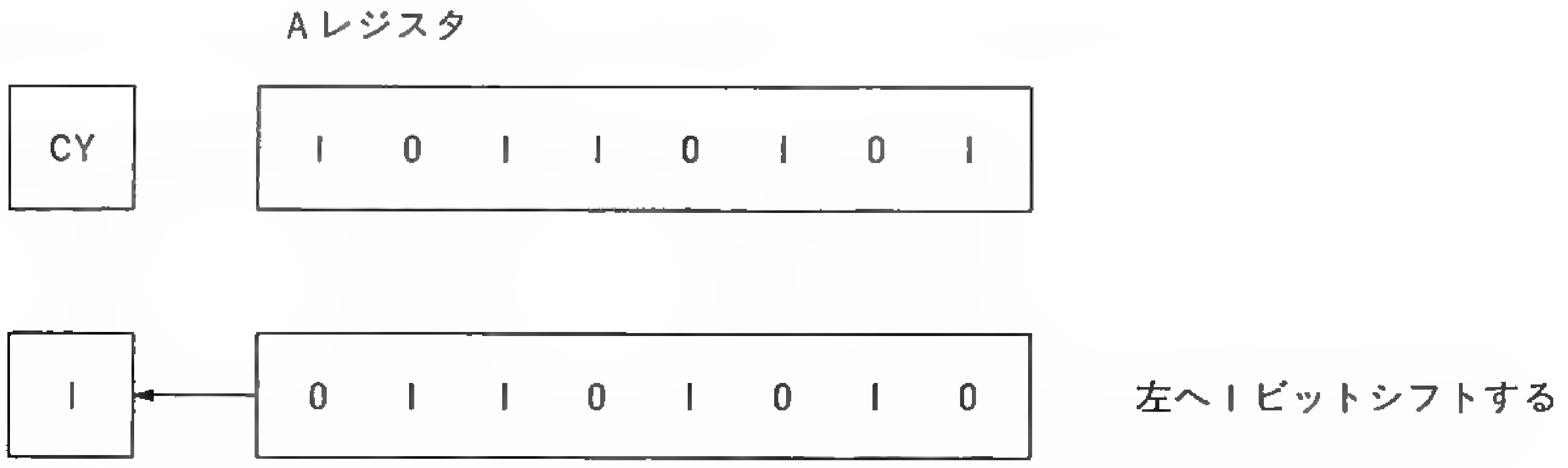
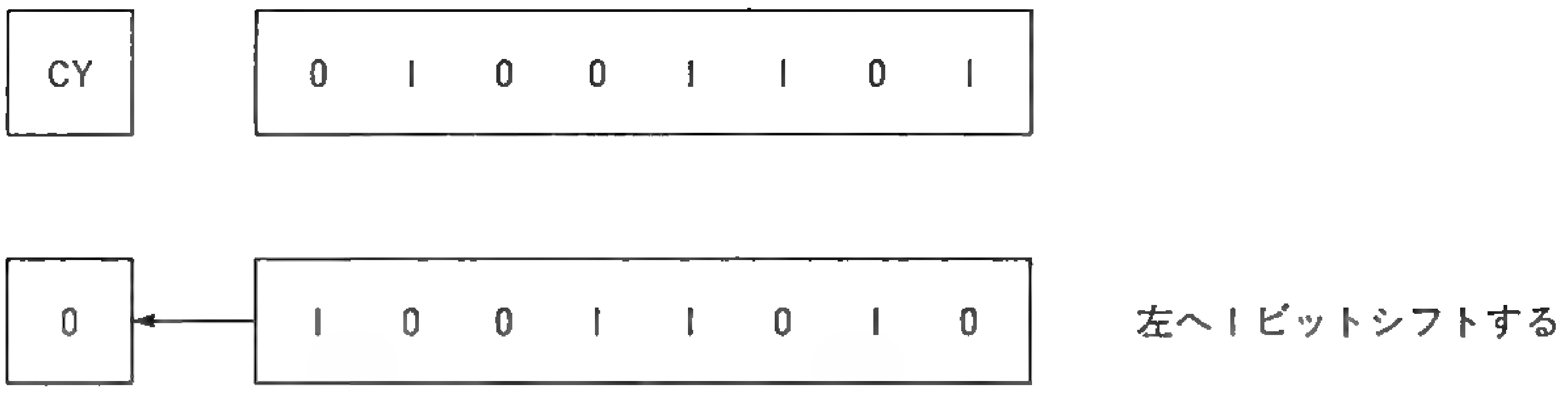

1 0 0 0 0 0 1 1
+ 1 0 0 0 1 1 0 0
1 0 0 0 0 1 1 1 1
桁上がり
CYフラグが1となる
0 1 0 0 0 0 1 1
+ 1 0 0 0 1 1 0 0
0 1 1 0 0 1 1 1 1
桁上がりがないのでCYフラグは0となる
桁借り
1 0 0 0 1 0 1 1 0
− 0 0 0 1 0 1 1 1
1 1 1 1 1 1 1 1
桁借りが生じたためCYフラグは1となる
1 0 0 1 0 1 1 0
− 0 0 0 1 0 1 1 1
0 1 1 1 1 1 1 1
桁借りが生じないのでCYフラグは0となる

図3.3 CYフラグの動作

CYフラグは，この他にもシフト演算のときに変化します．Aレジスタの8ビットのうちビット7が1であったとき，左に1ビットシフトするとどうなるでしょう．何もないとビット0からビット6の内容は，左へ1ビットずれただけで，最上位ビットの内容は消失してしまい，正しく2倍になりません．このような場合，ビット7の情報がそのままCYフラグにコピーされ，この場合CYフラグはセットされます(図3.4参照)．

また，論理演算の場合にもCYフラグは影響を受け，演算の結果にかかわらずリセットされます．


Aレジスタ
CY
1 0 1 1 0 1 0 1
1
0 1 1 0 1 0 1 0
左へ1ビットシフトする



キャリーを含めたシフト(ローテイト)の例

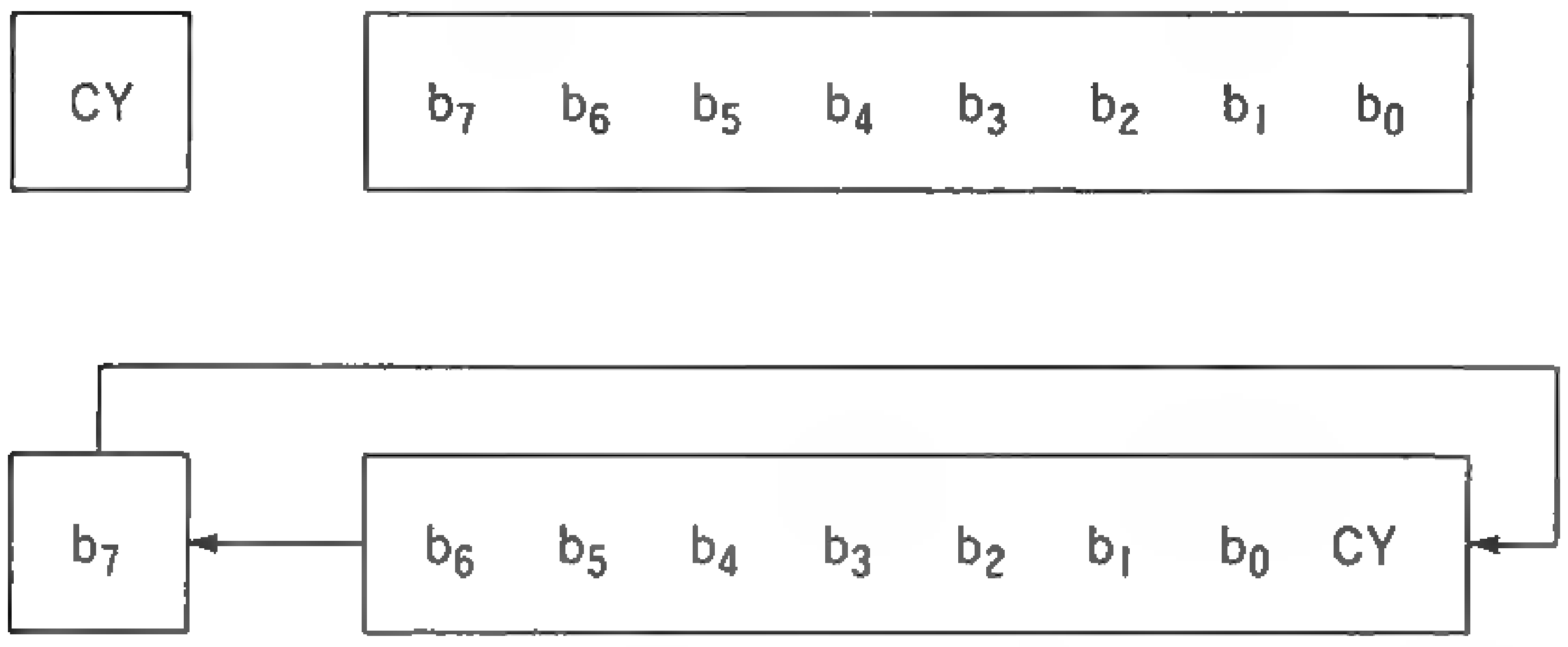

CY
b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0
b_7
b_6 b_5 b_4 b_3 b_2 b_1 b_0 CY

図3.4　シフト時のCYフラグ

３.３.２　Ｚフラグ

Ｚフラグは一般にゼロ・フラグと呼ばれます．このフラグは，演算の結果が０になったときセットされます．演算の結果が０になるというのはどのような場合かというと，減算を行って0になる(45－45＝０)場合，あるいは加算の結果Ａレジスタの内容が０になりキャリーがセットされる(つまり答えがちょうど$256_{(10)}$になる)場合などがあります．注意しなければならないのは，Ｚフラグは演算の結果が０になったときにセットされる(つまり１となる)ということです．決して，演算結果がゼロになったときに０になるわけではないのです．

３.３.３　Ｓフラグ

Ｓフラグは，サイン･フラグまたは符号フラグと呼ばれます．Ｓフラグは，演算の結果必ずＡレジスタのMSBの状態をそのままコピーします．2.4.3で説明した補数をよく思い出してください．補数は，MSBが１か０かで正負を判断することができました．つまりＳフラグは，演算の結果が正か負であるかを表していると考えてかまいません．ただし，これが成立するのは，補数を扱った場合のみです．

３.３.４　Ｐ／Ｖフラグ

Ｐ／Ｖフラグは，別名パリティ／オーバフロー・フラグと呼ばれます．このフラグには，次の２つの働きがあります．１つはオーバフロー・フラグとしての動作，もう１つはパリティ・フラグとしての動作です．なお8080Ａでは，このフラグはパリティ・フラグとしての動作しかありません．

オーバフロー・フラグとして動作するのは，加算や減算といった算術演算を行った場合です．

オーバフローとは補数演算を行ったときに，Aレジスタのビット数，つまり8ビットで表現された補数がその限界を超えたことをいいます．もちろん補数演算でない場合は，ほとんど問題になることはありません．具体的には，演算結果が＋127を超えた場合，または－128より小さかった場合です．図3.5をみてください．(a)，(b)は，オーバフローを起こさず，P／Vフラグはリセットされます．(c)，(d)は，演算結果がオーバーフローしてP／Vフラグがセットされる場合です．

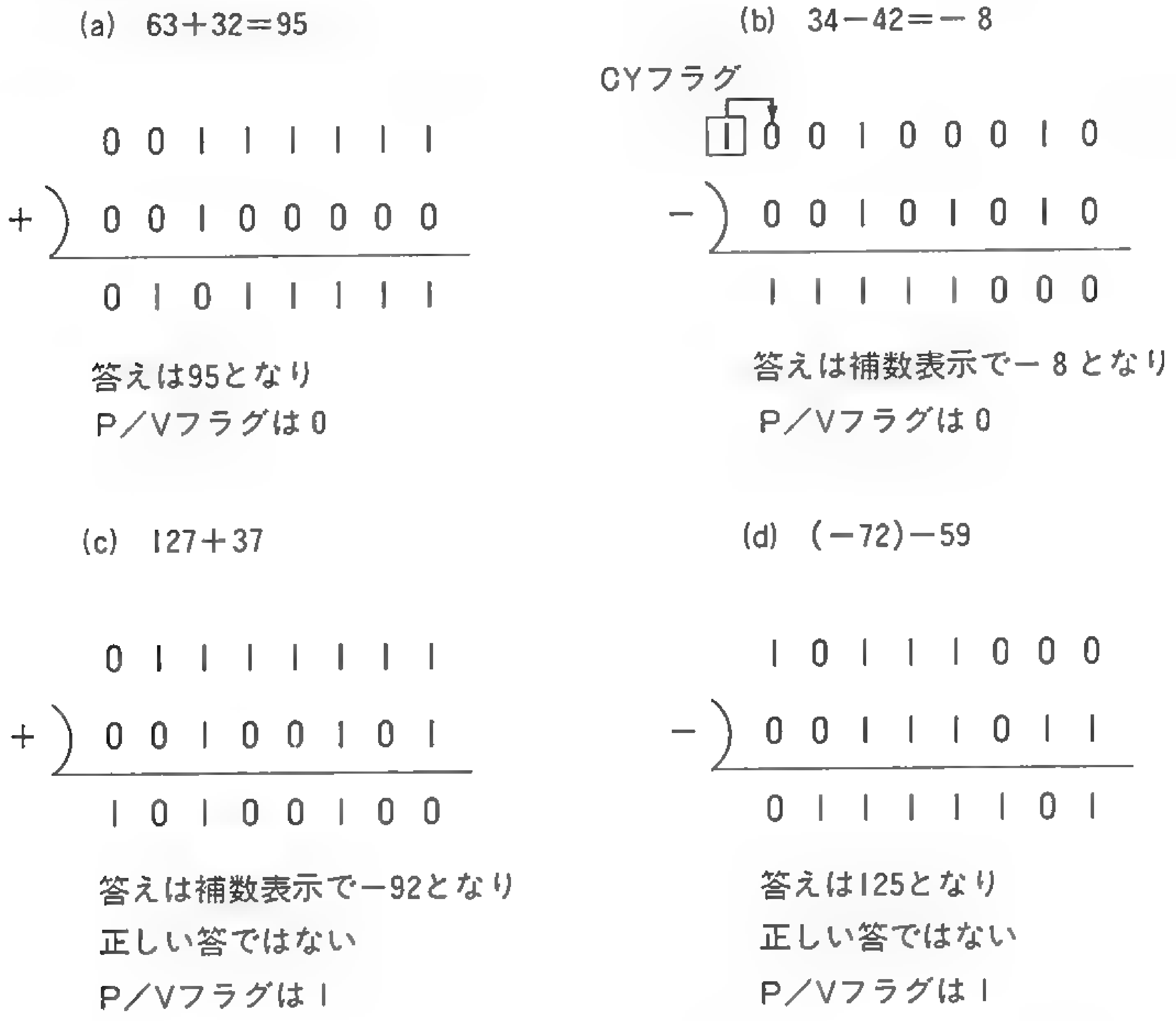

(a)　63＋32＝95
0 0 1 1 1 1 1 1
＋
0 0 1 0 0 0 0 0
0 1 0 1 1 1 1 1
答えは95となり
P／Vフラグは0
(b)　34－42＝－8
CYフラグ
1
0 0 1 0 0 0 1 0
－
0 0 1 0 1 0 1 0
1 1 1 1 1 0 0 0
答えは補数表示で－8となり
P／Vフラグは0
(c)　127＋37
0 1 1 1 1 1 1 1
＋
0 0 1 0 0 1 0 1
1 0 1 0 0 1 0 0
答えは補数表示で－92となり
正しい答ではない
P／Vフラグは1
(d)　(－72)－59
1 0 1 1 1 0 0 0
－
0 0 1 1 1 0 1 1
0 1 1 1 1 1 0 1
答えは125となり
正しい答ではない
P／Vフラグは1

図3.5　オーバフロー・フラグとしてのP／Vフラグの動作

パリティ・フラグとして動作するのは，論理演算を行った場合なのですが，パリティ・フラグの説明の前にパリティという言葉について説明します．パリティは，一般にはデータのチェックのために補助的に用いられます．パリティには奇数パリティと偶数パリティの2種類があり，それぞれ次のように動作します．

(1)　**奇数パリティ**……データ中の1が偶数個であればパリティは1，奇数個であれば0となる
(2)　**偶数パリティ**……データ中の1が奇数個であればパリティは1，偶数個であれば0となる

例として，アスキー・コードのパリティを図3.6に示します．

アスキー・コード	奇数パリティ	偶数パリティ
B：1 0 0 0 0 1 0	1	0
C：1 0 0 0 0 1 1	0	1

図3.6　アスキー・コードのパリティ

μPD780のP／Vフラグはデータだけをみた場合，データの中のセットされたビットが偶数であったときにセット，奇数であったときにリセットされます．

```
          10101001                  10101001
AND  )    01101011        OR  )     01101011
     ─────────────────         ─────────────────
          00101001                  11101011

         1の数が奇数               1の数が偶数
         P／Vフラグ…0              P／Vフラグ…1
```

図3.7　パリティ・フラグとしてのP／Vフラグの動作

3.3.5　Hフラグ，Nフラグ

Hフラグは，ハーフキャリー・フラグと呼ばれ，CYフラグがレジスタの最上位ビットの桁上がり，桁借りを表現していたのに対し，Hフラグは上位4ビットから下位4ビットへの桁借り，もしくは下位4ビットから上位4ビットへの桁上がりが生じたときセットされ，そうでない場合リセットされます．

Nフラグは加／減算フラグと呼ばれ，減算命令を行ったときにセットされ，加算命令を行ったときにリセットされます．

これら2つのフラグは，BCD演算を行うときに重要となりますが，プログラムの上での条件判断には利用されません．詳しくは第6章のBCD演算のところでもう一度説明します．

レジスタの役割とフラグの動作を一通りかけ足で説明しましたので，実感がつかめない方もいると思います．これらのフラグがどの命令によって，どのような影響を受けるかという具体的な対応と条件判断の利用については第5章で説明しますので，この節ではフラグの種類と役割について覚えていただければ結構です．

3.4　マシン語とニーモニック

第1章で，いきなりマシン語の内容をダンプしたのを覚えていますか．マシン語は，すべて16進数（根本的には2進数）で表現されていました．しかし，これでは誰がみても数字と文字の並びにすぎず，余程の人でない限りこれをすぐに理解することはできません．そこで考え出されたのがニーモニック（Mnemonic）です．ニーモニックは，マシン語を人間にとってもう少しわかりやすくするための1つの方法といえます．マシン語は16進数でしたが，ニーモニックは意味のある言葉を略したものです．例えば，Aレジスタの内容に，Bレジスタの内容を加えて結果をAレジスタに入れるという命令があります．これをマシン語とニーモニックでは次のように表現されます．

マシン語	ニーモニック
80	ADD　A，B

ADDは〝加える〟という意味の英語ですし，次のA，BはそれぞれAレジスタ，Bレジスタを表します．

そして，このニーモニックによって表される言語がアセンブリ言語(Assembler Language)です．

1つのニーモニックは，必ず1つのマシン語の命令と対応します．アセンブリ言語とニーモニックの詳しい説明は第5章で行います．

3.5　アセンブラと逆アセンブラ

前項で，「ニーモニックは，マシン語と1対1で対応しており，マシン語よりわかりやすい」と述べました．従ってマシン語のプログラムを開発する場合，一度アセンブリ言語で書き，それをマシン語に変換します．さて，このアセンブリ言語で書かれたプログラム(ソース・プログラム)をマシン語のプログラム(オブジェクト・プログラム)に変換する手段についてですが，これには次の2つの方法があります．

1つは，ソース・プログラムのニーモニックをマシン語↔ニーモニック対照表を調べながら，自分で1つずつマシン語の命令に置き換える方法です．この作業をハンドアセンブル（Hand Assemble）といいます．この方法は特別な設備はいっさい必要ありません．紙と鉛筆とマシン語↔ニーモニック対照表と時間さえあれば誰にでもできます．

もう1つの方法は，アセンブラ(Assembler)と呼ばれるプログラムを利用する方法です．これは，プログラムによってソース・プログラムをオブジェクト・プログラムに自動的に変換するものです．このようにアセンブラ・プログラムでオブジェクト・プログラムを生成することを，アセンブルするといいます．

■ハンドアセンブルの長所

- ●特別な設備はいっさい不要．よって資金がいらない

■アセンブラを利用する長所

- ●短時間で変換ができる
- ●変換の際の誤りがない

■ハンドアセンブルの短所

- ●短いプログラムならよいが，長いプログラムだと莫大な時間と労力がかかる．また誤りも多い

■ アセンブラを利用する短所

● 最低限，アセンブラとそれが動くシステムを必要とする

以上の長所，短所を考え合わせると，本格的にマシン語プログラムを作る場合には，やはりアセンブラがなければ難しいといえるでしょう．

ところで，ニーモニックからマシン語の命令が生成され，それらが1対1に対応するのなら，その逆ができてもよいはずです．つまり，マシン語のプログラムから，ニーモニックの集まりであるアセンブリ言語のプログラムを作ることができるはずです．これを逆アセンブルといいます．この方法もアセンブルと同様に，自分で行うハンド逆アセンブル(Hand Dis Assemble)，とプログラムに行わせる2つがあります．通常プログラムを利用して行う場合に，そのプログラムを逆アセンブラ(Dis Assembler)といいます．この逆アセンブラは，すでにでき上がったマシン語プログラムの解析などになくてはならないものです．

3.6　命令長による命令の区別

ここで命令長による命令の区別を考えます．命令長とは，1つの命令につき必要となるバイト数のことです．1バイトだけで命令が表現されるのであれば，2進数で00000000から11111111まで，16進数で00からFFまでの2^8=256種類の命令しか表現されないことになります．ところが実際には，256種類以上の命令があり，命令の中にはデータを含む命令もありますから，そのような命令は1バイトだけでは表現することはできません．当然2バイトや，3バイトが必要になります．μPD780には，1バイト命令から4バイト命令までの各種の命令があり，例えば，3.4で例に挙げたADD A, Bという命令は，マシン語にすると80Hとなり，1バイトで表されていました．

なぜ命令のバイト長の説明をしたかというと，似たようなニーモニックであっても，マシン語に変換すると全く異なったものとなって，命令長が変わることがよくあるからです(表3.3参照)．そして，あるニーモニックが何バイト命令であるかを理解することは，ハンドアセンブルでアドレス計算を行う際にとても重要となります．

	ニーモニック	マシン語
1バイト命令	LD A, B LD A, (HL)	78 7E
2バイト命令	LD A, 31H	3E 31
3バイト命令	LD A, (81F6H) LD A, (IX+6)	3A F6 81 DD 7E 06
4バイト命令	LD IX, 4321H	DD 21 21 43

表3.3　ニーモニックが同じでも命令長の異なる例

さて，表3.3に示した例は，すべてLDで始まっています．このLDは，ニーモニックで，操作(Operation)を表す部分であり，これをオペ・コード(Ope code)といいます．先程の，ADD A, Bは，ADDがオペ・コードです．これに対し，A, Bをオペランド(Operand)といい，操作するレジスタ，数値などを表します(Aの部分を第1オペランド，Bの部分を第2オペランドという)．いわば，オペ・コードが命令であって，オペランドは一種のデータと考えることができます．これは，BASICの命令とパラメータに似ています．マシン語のオペ・コード，オペランドの区別については，第5章および，付録の**インストラクション表**を参照してください．

3.7 μPD780の命令

μPD780の命令は以下に示すように大きく7種類に分けることができます．

(1) 転送命令	(5) 入／出力命令
(2) 演算命令	(6) 制御命令
(3) 分岐命令	(7) ブロック命令
(4) ビット命令	

表3.4 μPD780の命令

転送命令とは，データの転送，または交換を行う命令です．具体的には，Aレジスタの内容をBレジスタや，Cレジスタに移す命令，あるいはDEレジスタ・ペアとHLレジスタ・ペアの内容を交換するような命令があります．

演算命令とは，レジスタ間の加／減算，比較，論理演算，シフト，ローテイトなどの命令をいいます．

分岐命令とは，プログラムの流れを変える命令のことで，BASICでは，GOTO, GOSUB, RETURNに相当します．分岐命令には，大別して条件分岐命令と無条件分岐命令があり，無条件分岐命令は指定されたアドレスへ分岐(Jump)し，BASICにたとえると単なるGOTO文ということになります．条件分岐命令は，与えられた条件(この条件はフラグが1か0かによって決まる)が成立したとき分岐し，そうでない場合は次に進みます．BASICでは，IF～THEN〔行番号〕，IF～THEN GOSUB〔行番号〕，IF～THEN RETURNなどに相当します．

ビット命令は，ビットテスト命令，ビットセット命令，ビットリセット命令の三種類から成ります．ビットテスト命令は，指定されたレジスタ(またはメモリ)の指定されたビットが1であるか0であるかを調べる命令です．ビットセット命令は，そのビットを1にする(セット)命令で，ビットリセット命令はその逆で，0にする(リセット)命令です．

入／出力命令は，I/O装置とデータをやりとりする命令です．これには入力命令と，出力命令の2種類があります．BASICでは，それぞれINP，OUTに相当します．

制御命令とは，CPU自身を制御する命令で，動作の中断，割込みの許可，禁止などがあります．

ブロック命令は，いくつかの複合動作を1つの命令で行ってしまうもので，これには，ブロック転送，ブロック検索，ブロック入力，ブロック出力の4種類があります．

以上が，おおまかにみたμPD780の命令です．実際のプログラムは，これらの命令が複雑に関係し合って成り立っており，レジスタ，メモリ，I/Oポートの間でデータを直接操作するという細かな手続きが必要です．BASICプログラムのように，PRINT文だけで表示ができるとか，GOTO文だけでプログラムが動くといったこととは大きな隔たりがあります．

3.8　アルゴリズムとフローチャート

BASICで1＋1の計算を行う場合には次のように1行で記述できます．

```
PRINT   1＋1
```

ところがマシン語で，この計算を行う場合には次の3行が必要ですが，これだけでは計算を行っただけで，実行結果はどこへも出力されませんから実行結果を知りたい場合，求めた値を表示するプログラムを付け加えねばなりません．

```
LD      A，1
LD      B，1
ADD     A，B
```

このように，マシン語プログラムはきわめて原始的なものです．そのため，マシン語においては命令の手順が特に重要となってくるのです．皆さんはアルゴリズム（Algorithm）という言葉を，多少なりとも聞いたことがあると思います．例えば1から100までの和を求める計算機向き（人間向きではない）のアルゴリズムは次のようになります．

(1)　Aに0を代入する　A＝0
(2)　Bに1を代入する　B＝1
(3)　AにBとAを足したものを代入する　A＝B＋A
(4)　Bの値に1を足す　B＝B＋1
(5)　Bの値が101になったら終わりでAが答えとなる，そうでなければ
　(3)へ戻る

この例からわかるように，アルゴリズムという言葉は一連の計算手順を意味します．

アルゴリズムについては，すでにいろいろな研究がなされていますので，皆さんがマシン語でプログラムを作る際には，そのようなアルゴリズムについて述べてある本を参考にするとよいかと思います．

さて，アルゴリズムを，比較的わかりやすくする表現法に，フローチャート（Flow Chart）とい

うものがあります．先程の１から100までの和を求めるアルゴリズムをフローチャートで表現したのが図3.8です．

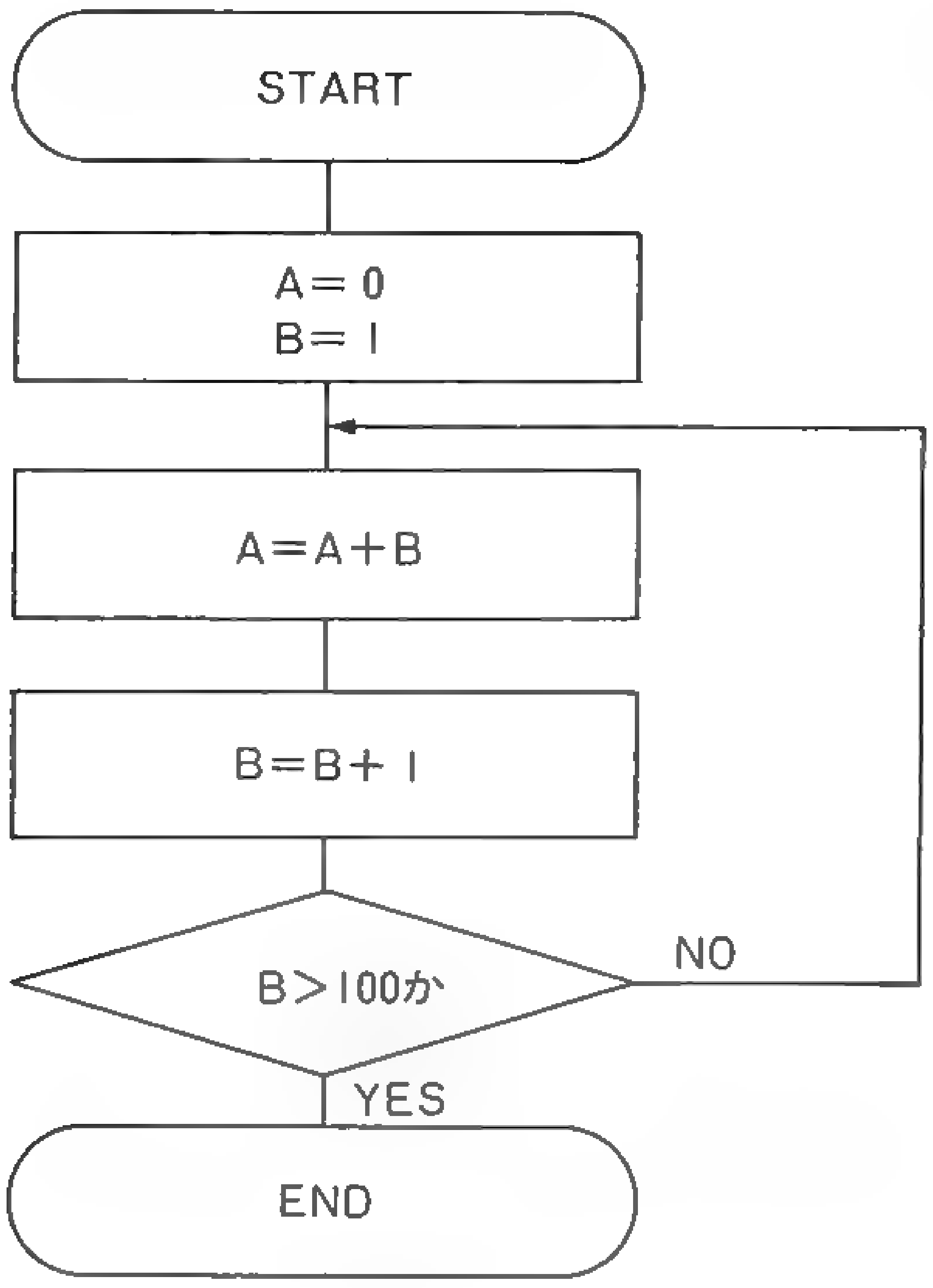


図3.8　1から100までの和を求めるアルゴリズムのフローチャート

一般に，フローチャートは，上から下へみるように表現されます．このフローチャートには，３つの記号とそれらを結ぶ線から成り立っています．

⬭：処理の開始および終了を表現する
▭：一連の処理を表現する
◇：判断を表す．この判断の結果によって行き先が変わる

フローチャートについては，JISによって詳しく決められています．上に示したのは，そのうちの代表的なものです．

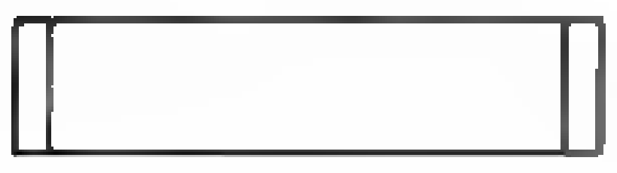
マシン語で複雑なプログラムを作成する場合には，ぜひフローチャートを作成してからプログラムを作成することをお勧めします．一度ですべてうまくいけばよいのですが，バグがあった場合には，フローチャートが大きな力を発揮します．

3.9　サブルーチンについて

BASICの場合にもサブルーチンの考え方がありました．マシン語の場合もこれと同じです．要するに何度も同じ処理を行う場合にメモリの節約となりますし，サブルーチン化することによって，各ルーチンの汎用性が増します．特にマシン語の場合は，サブルーチンに分岐する場合も実行時間にほとんど影響を与えませんのでかなり有効です．

BASICでサブルーチンに分岐する場合は，GOSUB文を使いましたが，マシン語には同様の働きをする命令に，CALL命令(ニーモニック表現)があり，また，RETURNに相当する命令にRETがあります．これらはそれぞれが，分岐命令と同様に無条件に実行するものと条件によって実行するものです(詳しくは第5章参照)．

さて，前項で説明したフローチャートで，サブルーチンの記述法について少し述べておきます．**図3.9**をみてください．手続き，AとBが2回ずつ出てきます．このような表現法をとることによって，サブルーチンの煩雑なフローチャートの繰り返しを避けることができます．

：あらかじめ定められた手続きを表現する(サブルーチン)

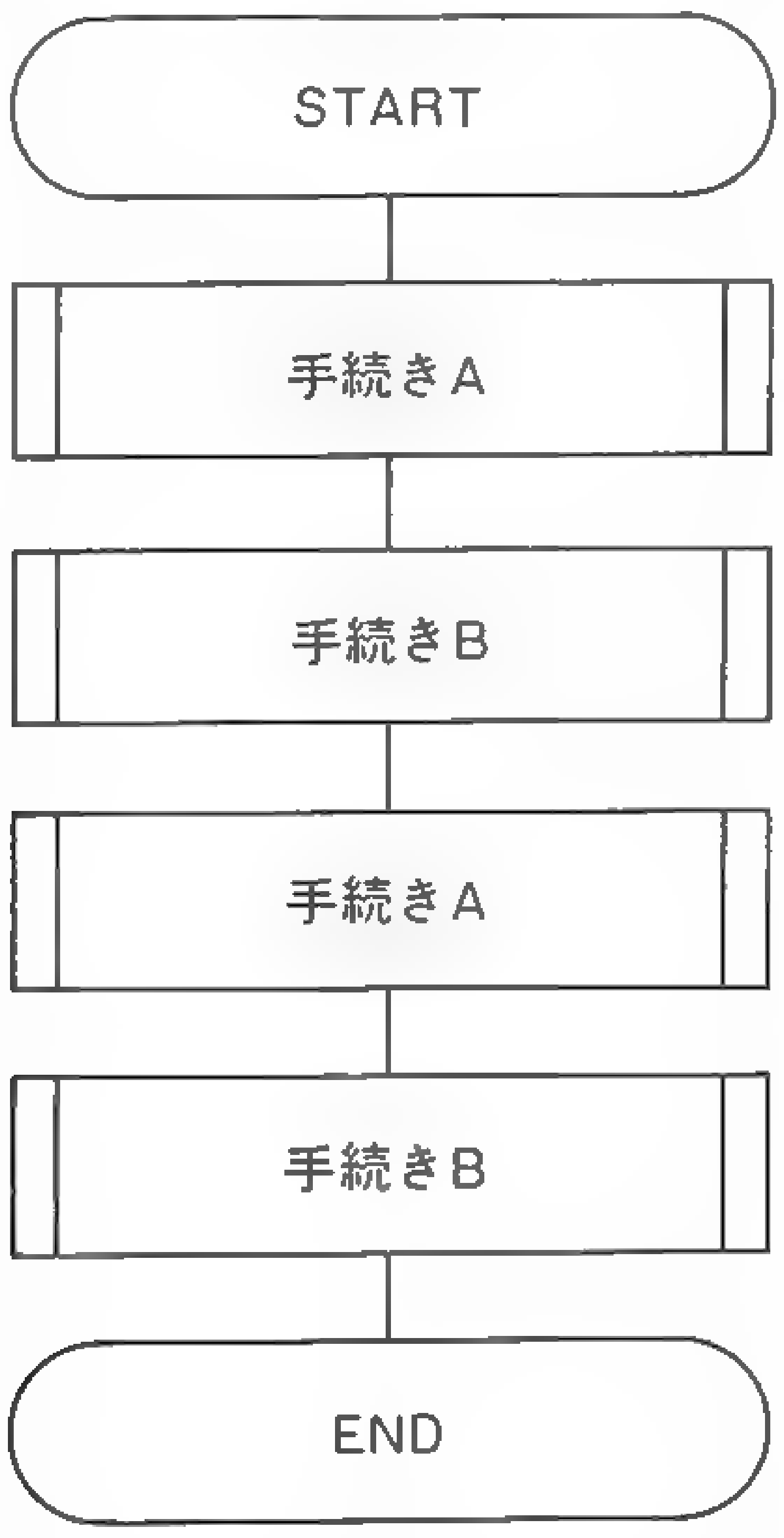


図3.9　サブルーチンのフローチャート

第4章　モニタ機能とアドレス

コンピュータの書籍，雑誌にはかなり多くのマシン語プログラムが掲載されていますので，皆さんも一度は目にしたことがあると思います．しかしそのプログラムの形式は，16進数の並びであったり，英字と16進数の混在で表されていますから，BASICのプログラムとは似ても似つかないものになっています．

皆さんご承知の通り，BASICのプログラム入力は，行番号，ステートメント，リターンキーの順に入力していけば，それはもうプログラムとしてコンピュータに記憶されてしまいます．では，行番号すら付いていないマシン語プログラムはどのように入力するのでしょうか．入力だけではありません，デバッグ，セーブ，ロード，実行とBASICでは普段何気なく行っていることも，マシン語ではBASICと少々異なった方法で行います．

本章ではマシン語プログラムを編集(入力，デバッグ，セーブ，ロード，実行等)するために，PC-8801に備わっているモニタ・コマンドを説明すると共に，BASICとリンクする際のマシン語プログラムのアドレスの設定について解説します．

4.1　モニタとは

モニタとは，一体何なのでしょう．よく，コンピュータに使用するテレビのようなスクリーン装置(CRT)をモニタといいます．また，テレビ局などが番組内容のレポートを提出してもらう視聴者の人たちも，モニタと呼ばれます．しかしながら，ここでいうモニタはこれらのモニタとは少々意味が違うようです．

モニタは英語ではmonitorと綴り，その意味を辞書で調べると，監視者，監視装置といった意味があり，前述した2種類のモニタはこれらの意味で使われているようです．それに対してコンピュータに関連して使われているモニタは〝マシン語プログラムを監視するもの〟という意味です．

4.1.1　モニタの機能

一般にパーソナル・コンピュータのモニタ機能は貧弱なものが多く，後で述べる最低限必要な機能しか備えていないものがほとんどです．しかし，PC-8801のモニタ機能は従来の機種とは比較にならない程多くの機能を備えており，ユーザのマシン語プログラミングの大きな手助けとなっています．

モニタとしての機能は，最低限次の3つが必要です．

(1)　メモリの内容を調べる
(2)　メモリの内容を書き換える
(3)　ユーザが作成したプログラムを実行する

N$_{88}$-BASICのモニタは，上記の3つを充分に満たしていることはいうまでもありませんが，その他にも多くの機能を持っています．

4.2　N$_{88}$-BASICのモニタ

モニタ・モードに入るためには，BASICのダイレクトステートメントで次のように入力します．

```
MON ↲
```

すると，次のようなプロンプトが表示され，コマンド入力待ちとなります．

```
h]
```

さて，プロンプトは入力を待ち続けています．適当なキーを入力してみますか？どうです，運よく何か返事のもらえた人もいるかもしれませんが，おそらく多くの人はモニタに〝ピー〟と怒られたのではないでしょうか．それではモニタ入力の決まりを覚えてしまいましょう．

4.2.1　モニタで使用する文字

まずモニタで使用できる数字ですが，これには16進数モードと8進数モードがあり，これらの切り換えは後述するBコマンドによって行います，本書においては16進数表記で統一します．16進数モードにおいては，数値はすべて16進数とみなされますので，通常16進数表記に付けるHの文字は付けてはいけません．また10進数で表現された数は，16進数に変換する必要があります．

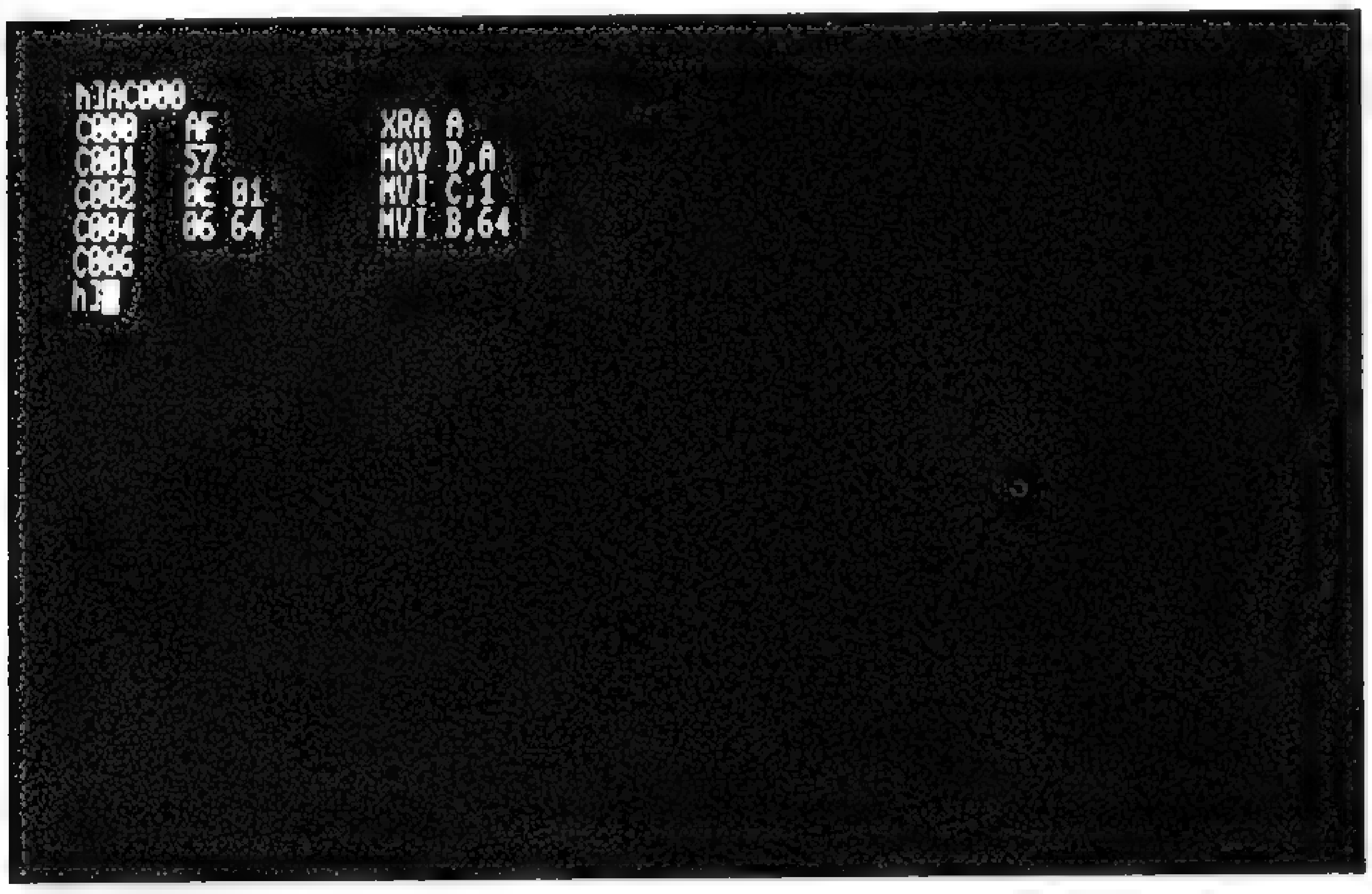
コマンドなどを入力する場合，アルファベットの大文字，小文字は全く同様に扱われますが，以下の説明ではすべて大文字を用いることにします．

4.2.2　コマンドの説明

▶ A（アセンブル）

書式 Assss　　ssss：スタート・アドレス

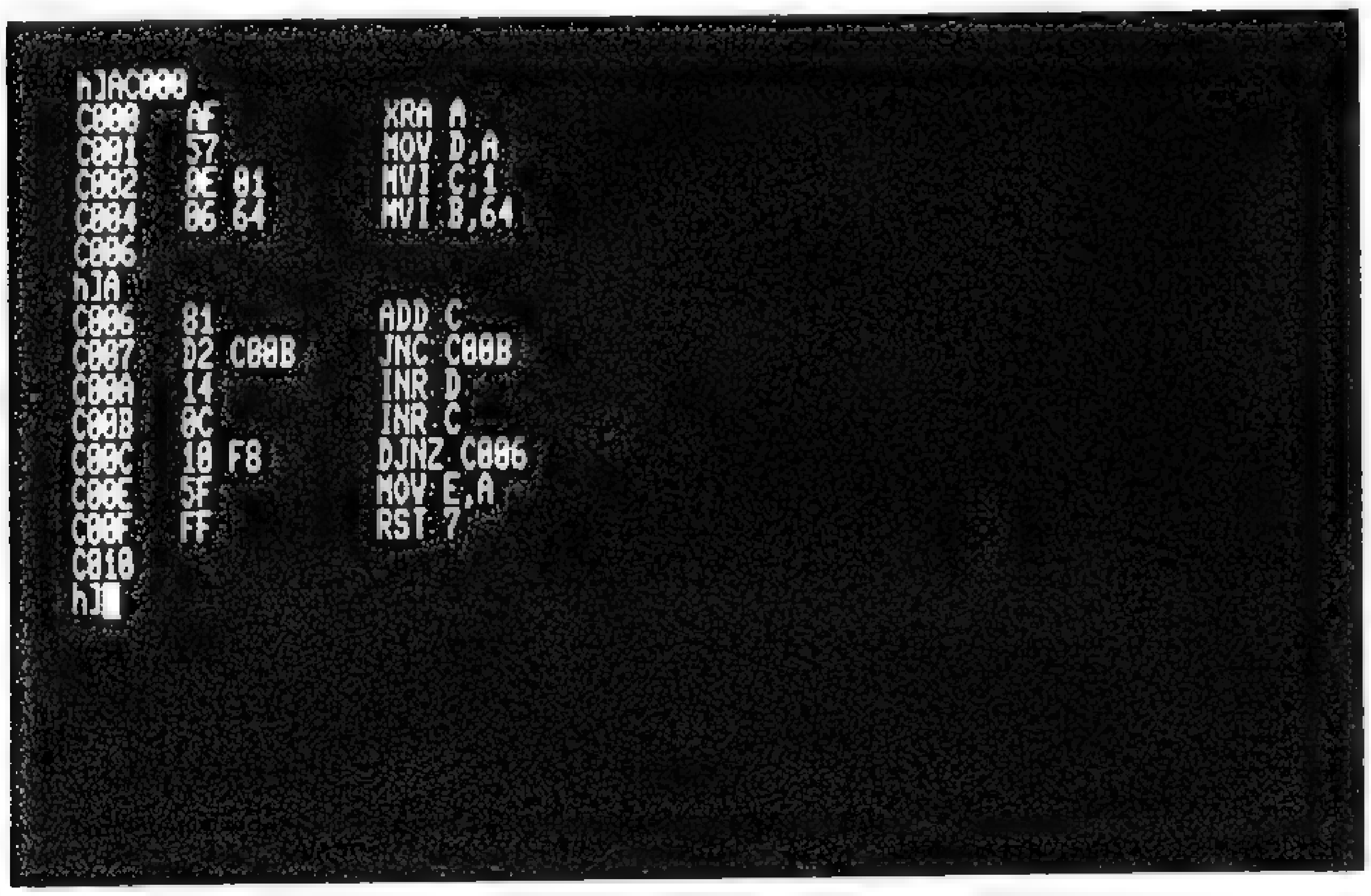
1行ごとにアセンブルを行い，そのオブジェクト・コードをアドレスssssからロードしていきます．この場合のニーモニックは，インテル社の8080Aのニーモニックに準拠しており，μPD780のすべての命令はサポートされていません．



```
h]AC000
C000   AF          XRA A
C001   57          MOV D,A
C002   0E 01       MVI C,1
C004   06 64       MVI B,64
C006
h]
```

Aコマンドの実行例
スタート・アドレスのみ指定
した場合

ssssを指定しない場合は，直前のAコマンドが終了したアドレスの次のアドレスとなります．



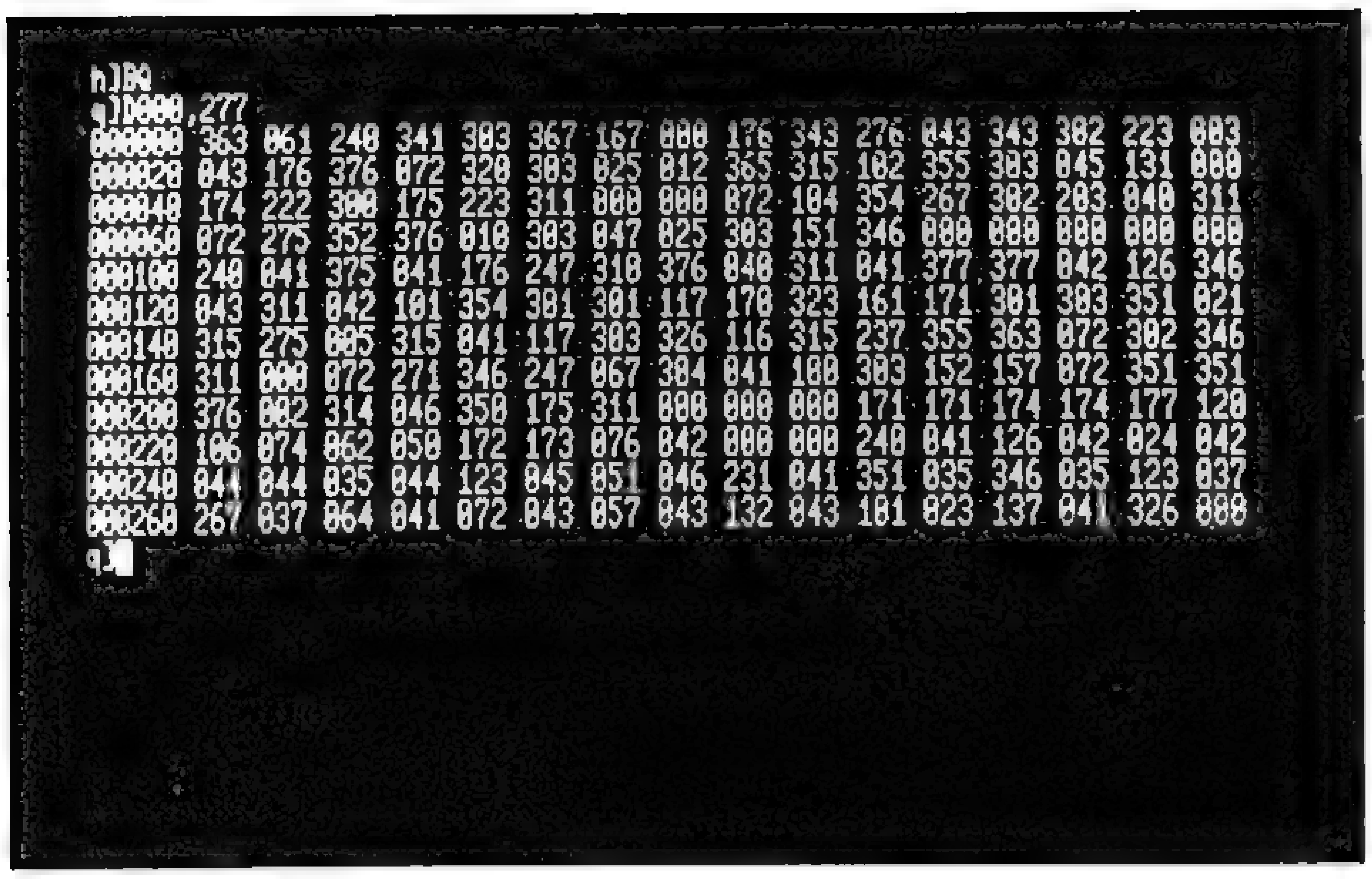
```
h]AC000
C000   AF          XRA A
C001   57          MOV D,A
C002   0E 01       MVI C,1
C004   06 64       MVI B,64
C006
h]A
C006   81          ADD C
C007   D2 C00B     JNC C00B
C00A   14          INR D
C00B   0C          INR C
C00C   10 F8       DJNZ C006
C00E   5F          MOV E,A
C00F   FF          RST 7
C010
h]
```

Aコマンドを終了するには，何も入力せずにリターンキーを押します．

▶ B（ベース）

書式 BH／BQ

Hを指定した場合扱える数値を16進数にします．このときのプロンプトはh］となり，BQの命令は扱える数値を8進数にします．このときのプロンプトはq］となります．



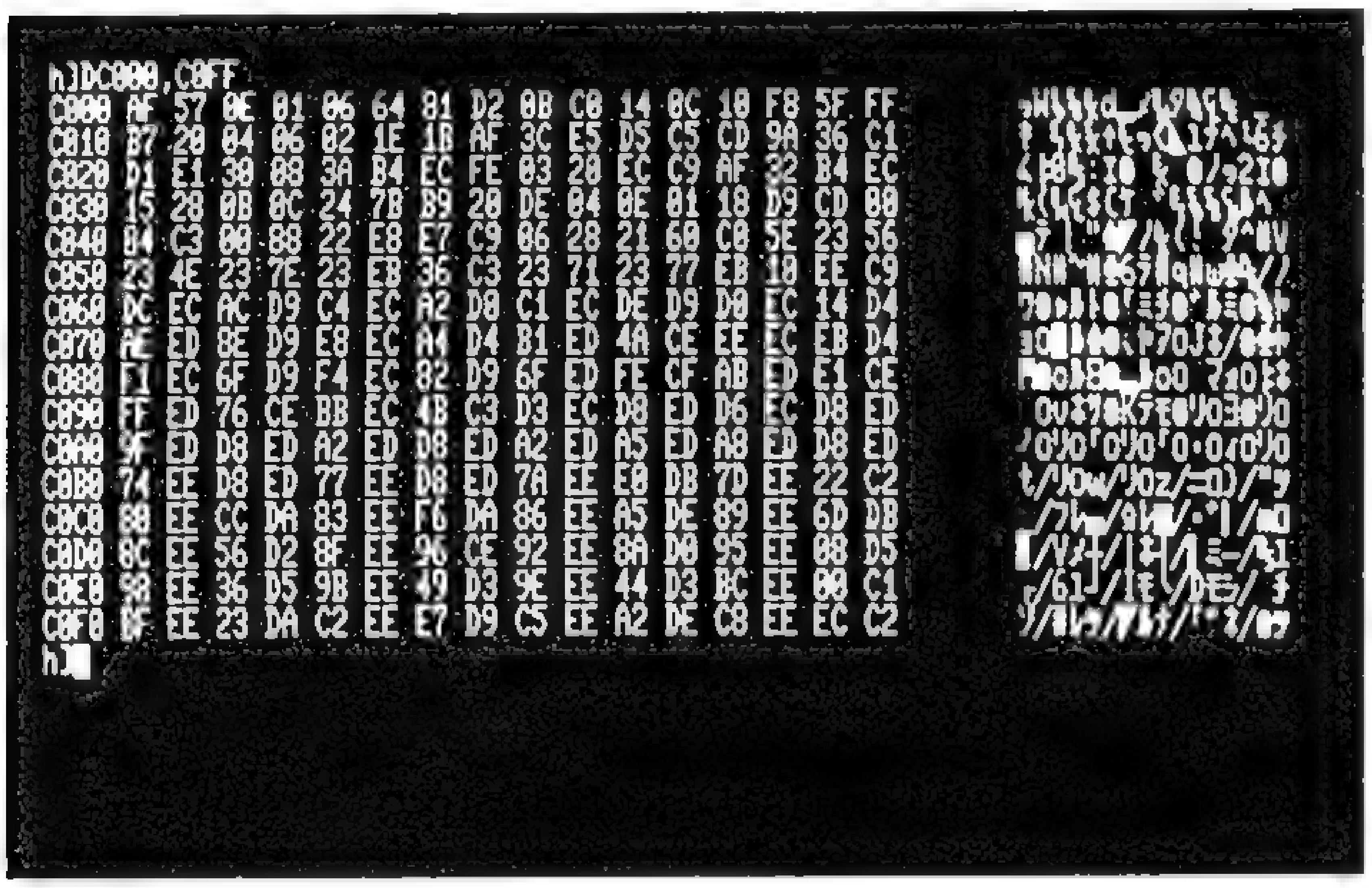
```
h]BQ
q]D000,277
000000 363 061 240 341 303 367 167 000 176 343 276 043 343 302 223 003
000020 043 176 376 072 320 303 025 012 365 315 102 355 303 045 131 000
000040 174 222 300 175 223 311 000 000 072 104 354 267 302 203 040 311
000060 072 275 352 376 010 303 047 025 303 151 346 000 000 000 000 000
000100 240 041 375 041 176 247 310 376 040 311 041 377 377 042 126 346
000120 043 311 042 101 354 301 301 117 170 323 161 171 301 303 351 021
000140 315 275 005 315 041 117 303 326 116 315 237 355 363 072 302 346
000160 311 000 072 271 346 247 067 304 041 100 303 152 157 072 351 351
000200 376 002 314 046 350 175 311 000 000 000 171 171 174 174 177 120
000220 106 074 062 050 172 173 076 042 000 000 240 041 126 042 024 042
000240 041 044 035 044 123 045 051 046 231 041 351 035 346 035 123 037
000260 267 037 064 041 072 043 057 043 132 043 101 023 137 041 326 000
q]
```

HはHexadecimal, QはOctal 指定

▶ D（ダンプ）

書式 Dssss, eeee　　eeee：エンド・アドレス

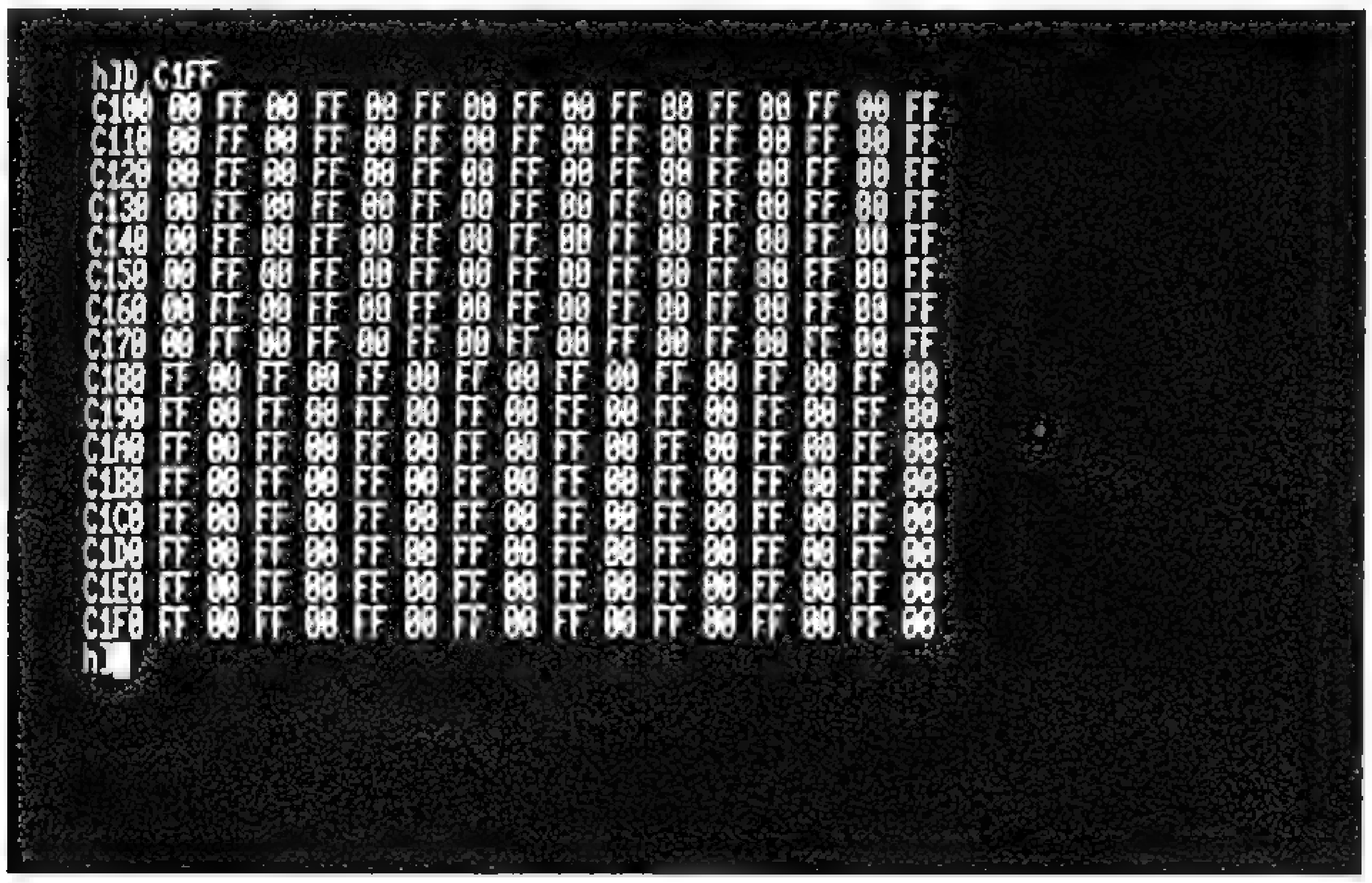
アドレスssssからeeeeまでのメモリ内容をダンプします．16進表示の右側には，メモリの内容に対応するPC-8801のキャラクタが表示されます．



```
h]DC000,C0FF
C000 AF 57 0E 01 06 64 81 D2 0B C0 14 0C 10 F8 5F FF
C010 B7 20 04 06 02 1E 1B AF 3C E5 D5 C5 CD 9A 36 C1
C020 D1 E1 30 08 3A B4 EC FE 03 20 EC C9 AF 32 B4 EC
C030 15 28 0B 0C 24 7B B9 20 DE 04 0E 01 18 D9 CD 00
C040 04 C3 00 08 22 E8 E7 C9 06 28 21 60 C0 5E 23 56
C050 23 4E 23 7E 23 EB 36 C3 23 71 23 77 EB 18 EE C9
C060 DC EC AC D9 C4 EC A2 D8 C1 EC DE D9 D0 EC 14 D4
C070 AE ED 8E D9 E8 EC A4 D4 B1 ED 4A CE EE EC EB D4
C080 F1 EC 6F D9 F4 EC 82 D9 6F ED FE CF AB ED E1 CE
C090 FF ED 76 CE BB EC 4B C3 D3 EC D8 ED D6 EC D8 ED
C0A0 9F ED D8 ED A2 ED D8 ED A2 ED A5 ED A8 ED D8 ED
C0B0 74 EE D8 ED 77 EE D8 ED 7A EE E0 DB 7D EE 22 C2
C0C0 88 EE CC DA 83 EE F6 DA 86 EE A5 DE 89 EE 6D DB
C0D0 8C EE 56 D2 8F EE 96 CE 92 EE 8A D0 95 EE 08 D5
C0E0 98 EE 36 D5 9B EE 49 D3 9E EE 44 D3 BC EE 00 C1
C0F0 BF EE 23 DA C2 EE E7 D9 C5 EE A2 DE C8 EE EC C2
h]
```

256バイト以上のダンプはスクロールされる

書式 D, eeee

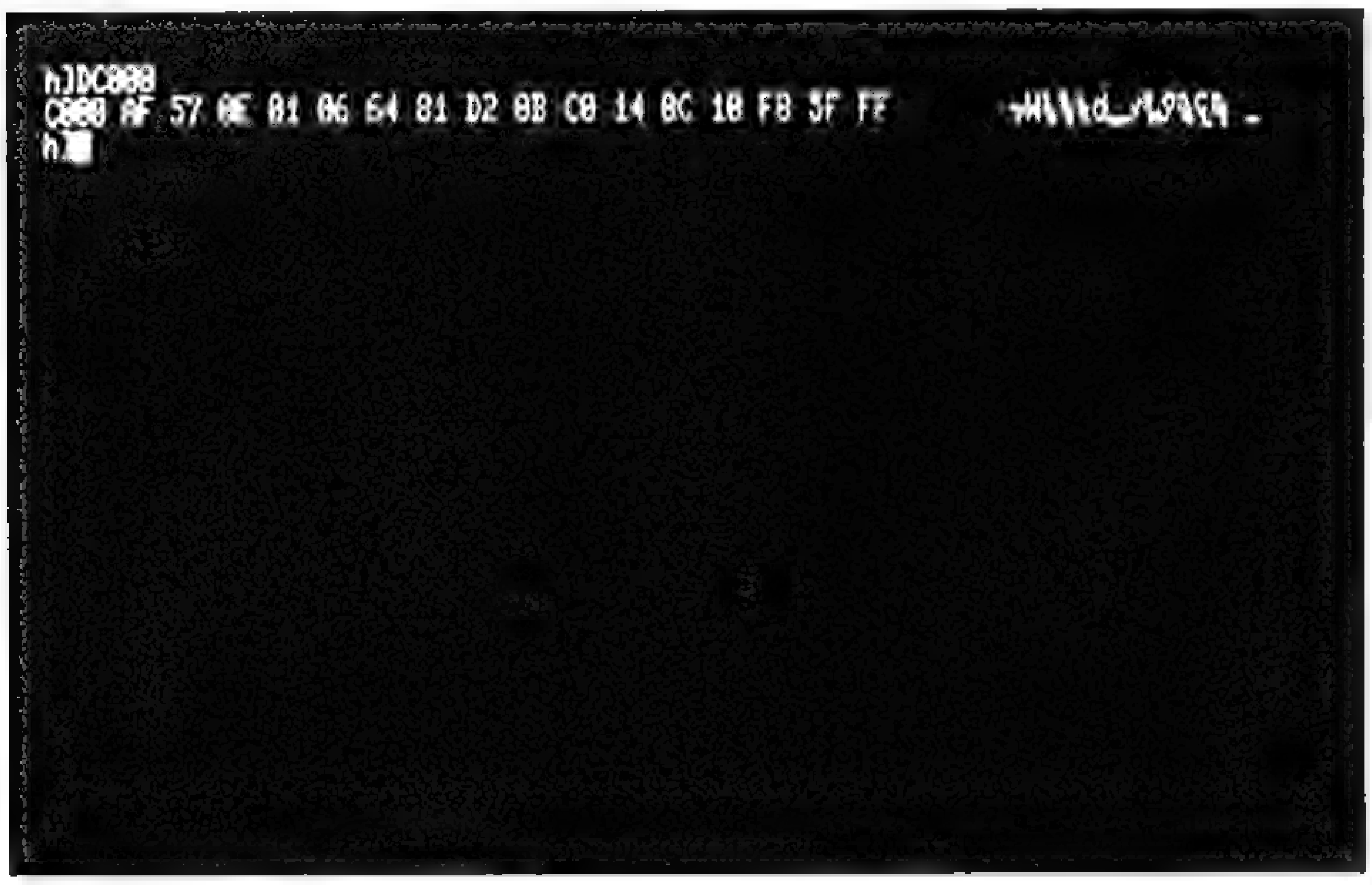


```
h]D,C1FF
C100 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
C110 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
C120 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
C130 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
C140 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
C150 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
C160 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
C170 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
C180 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
C190 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
C1A0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
C1B0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
C1C0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
C1D0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
C1E0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
C1F0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
h]
```

前回のＤコマンド表示したエンド・アドレスの次のアドレスからアドレスeeeeまでの内容を表示します.

書式 Dssss

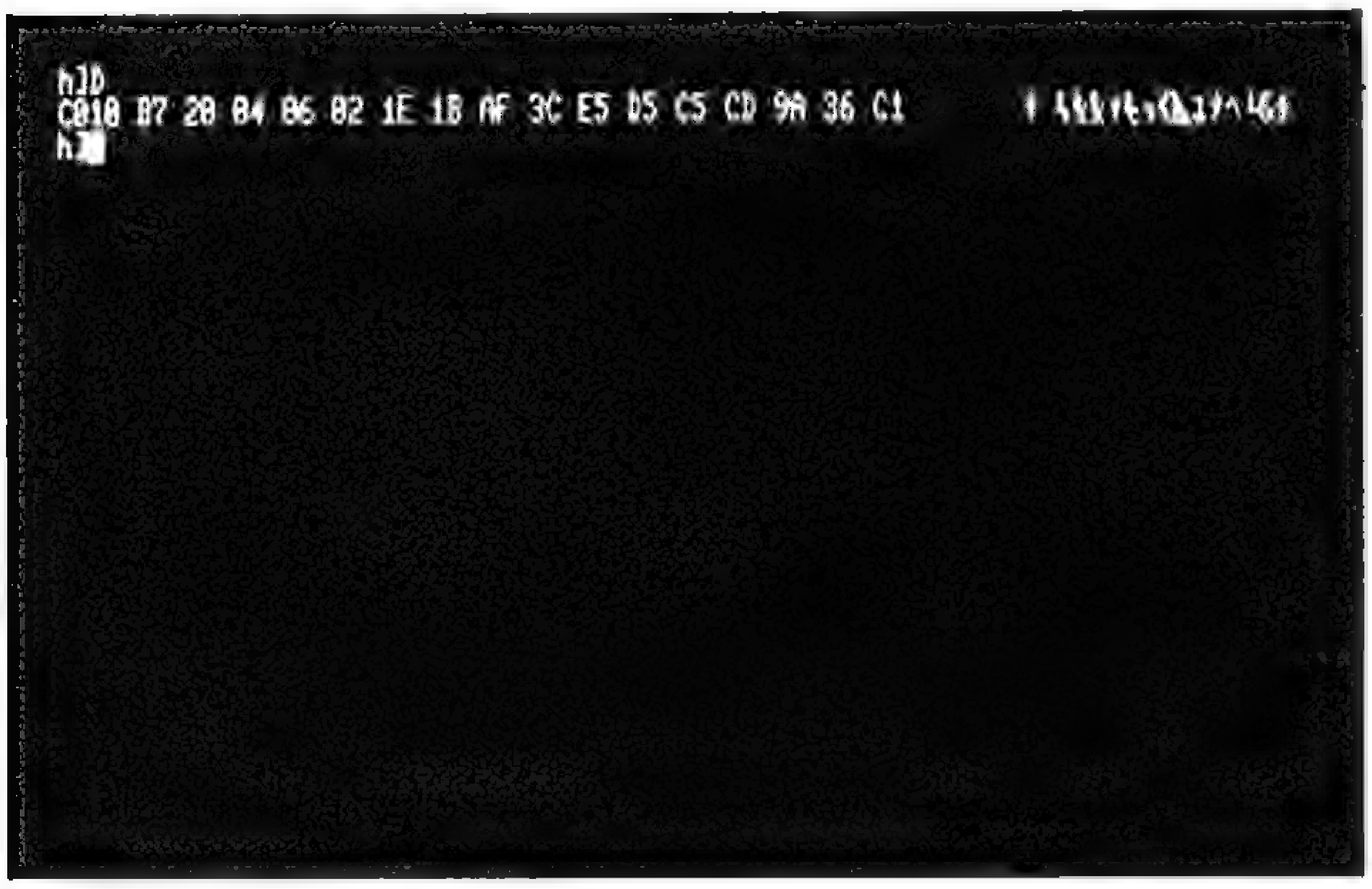
ssssから始まる16バイト分の内容を表示します.



```
h]DC000
C000 AF 57 0E 01 06 64 01 D2 0B C0 14 0C 18 F8 3F FF
h]
```

書式 D

前回のＤコマンド表示したエンド・アドレスの次のアドレスから16バイト分の内容を表示します.

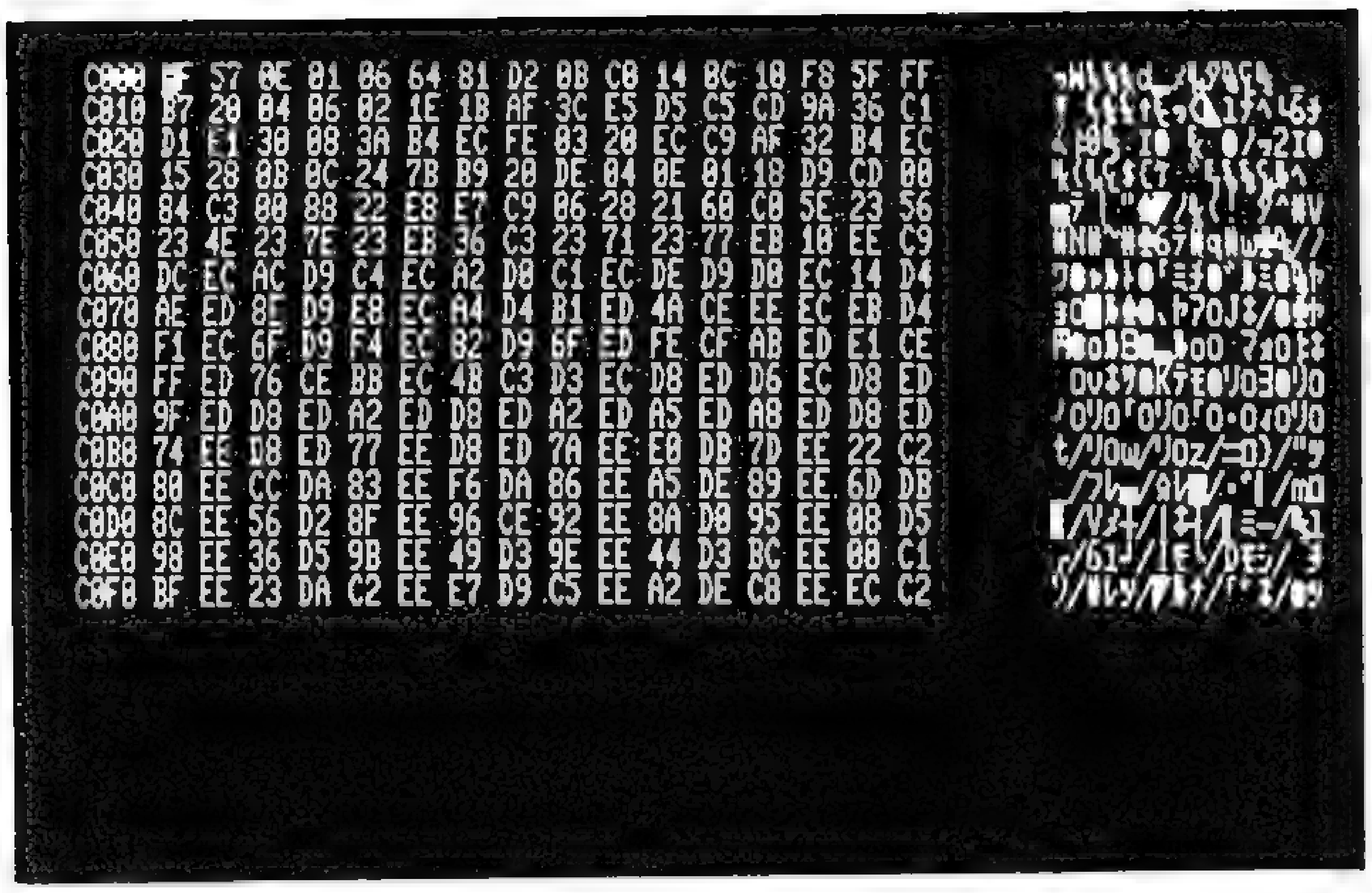


```
h]D
C010 B7 20 04 06 02 1E 1B AF 3C E5 D5 C5 CD 9A 36 C1
h]
```

▶ E（エディット）

書式 Essss

Ｅコマンドは，スクリーンエディタ機能を活かしたメモリ変更コマンドです.アドレス ssssから始まる256バイト（１行80字の場合．40字の場合は128バイト）を画面をクリアして表示します．その後，カーソル・コントロールキーと ROLL UP ，ROLL DOWN を使って，メモリの内容を変更したいところへカーソルを移動させ，メモリの内容を変更します．16進数にあたるキーを押すと，それがそのままメモリの内容となります.



```
C000 FF 57 0E 01 06 64 81 D2 0B C8 14 0C 10 F8 5F FF
C010 B7 20 04 06 02 1E 1B AF 3C E5 D5 C5 CD 9A 36 C1
C020 D1 E1 30 08 3A B4 EC FE 03 20 EC C9 AF 32 B4 EC
C030 15 28 0B 0C 24 7B B9 20 DE 04 0E 01 18 D9 CD 00
C040 84 C3 00 88 22 E8 E7 C9 06 28 21 60 C0 5E 23 56
C050 23 4E 23 7E 23 EB 36 C3 23 71 23 77 EB 10 EE C9
C060 DC EC AC D9 C4 EC A2 D0 C1 EC DE D9 D0 EC 14 D4
C070 AE ED 8E D9 E8 EC A4 D4 B1 ED 4A CE EE EC EB D4
C080 F1 EC 6F D9 F4 EC 82 D9 6F ED FE CF AB ED E1 CE
C090 FF ED 76 CE BB EC 48 C3 D3 EC D8 ED D6 EC D8 ED
C0A0 9F ED D8 ED A2 ED D8 ED A2 ED A5 ED A8 ED D8 ED
C0B0 74 EE D8 ED 77 EE D8 ED 7A EE E0 DB 7D EE 22 C2
C0C0 80 EE CC DA 83 EE F6 DA 86 EE A5 DE 89 EE 6D DB
C0D0 8C EE 56 D2 8F EE 96 CE 92 EE 8A D0 95 EE 08 D5
C0E0 98 EE 36 D5 9B EE 49 D3 9E EE 44 D3 BC EE 00 C1
C0F0 BF EE 23 DA C2 EE E7 D9 C5 EE A2 DE C8 EE EC C2
```

スタート・アドレスにC000H
を指定した場合

書式 E

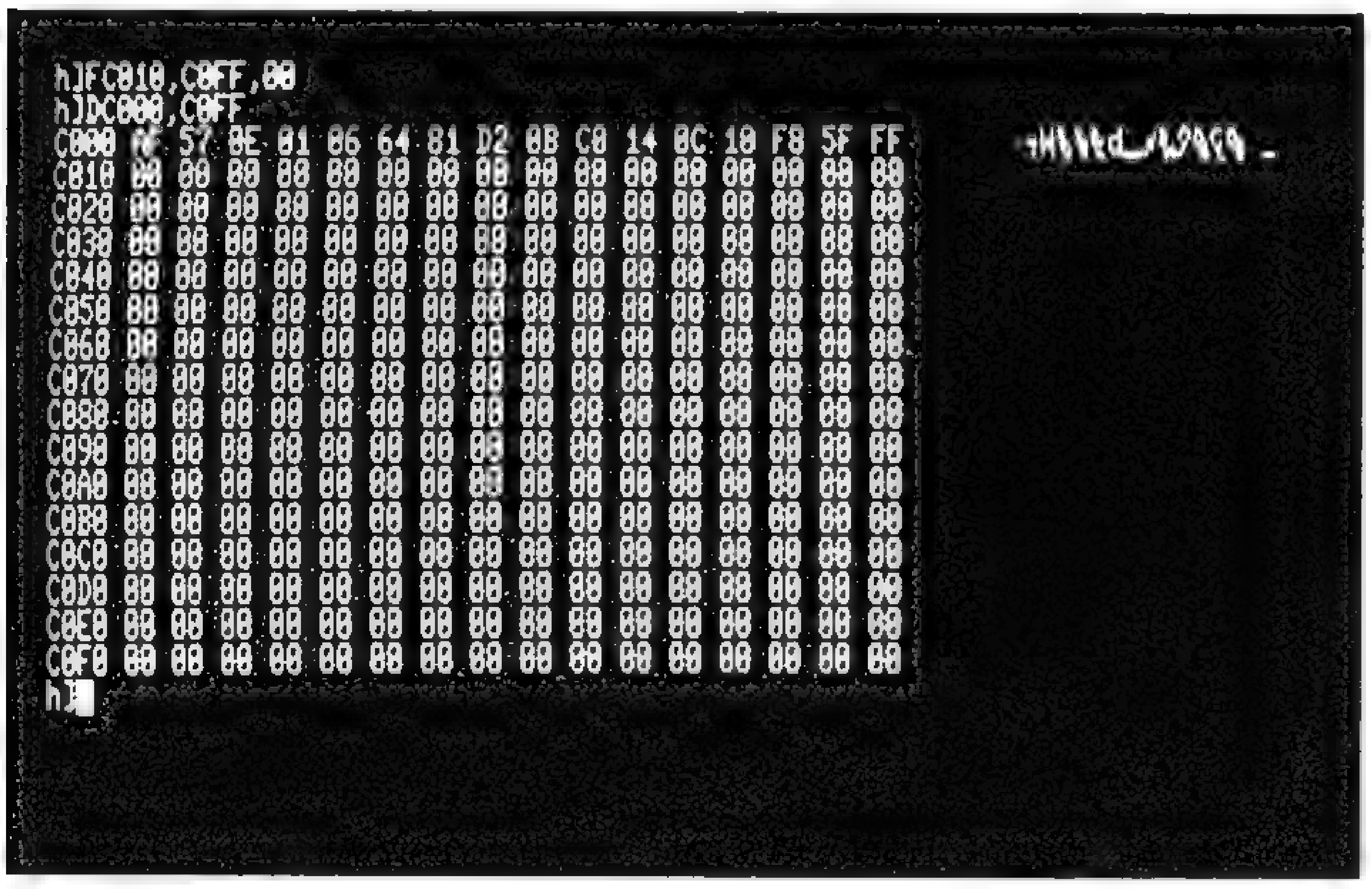
アドレスを省略した場合には最後にEコマンドで内容を変更したメモリのアドレスがスタート・アドレスとなります．

なお，一度Eコマンドで設定した画面では数値入力だけではFFバイト以上の変更はできないので，さらに続ける場合には，［ROLL UP］，［ROLL DOWN］でアドレスの移動が必要です．

▶ F（メモリ・フィル）

書式 Fssss, eeee, cc　　cc：データ

アドレスssssからeeeeまでのメモリ全部にデータccを書き込みます．



```
hJFC010,C0FF,00
hJDC000,C0FF
C000 AF 57 0E 01 06 64 81 D2 0B C8 14 0C 10 F8 5F FF
C010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
hJ
```

C010H〜C0FFHまでをデータ00Hでうめる

▶ G（ゴー）

書式 Gssss, bbbb／Gssss, bbbb, b′b′b′b′　　bbbb：ブレークポイント1
b′b′b′b′：ブレークポイント2

1つまたは2つのブレークポイント（bbbb, b′b′b′b′）を設けて，アドレスssssから，リアルタイムで被テスト・プログラムを実行します．

書式 G, bbbb／G, bbbb, b′b′b′b′

スタート・アドレスが現在のプログラム・カウンタ（Xコマンドで表示，XPコマンドで変更可能）であること以外は上記と同様です．

書式 Gssss

ブレークポイントを設けずに，アドレスssssから実行します．

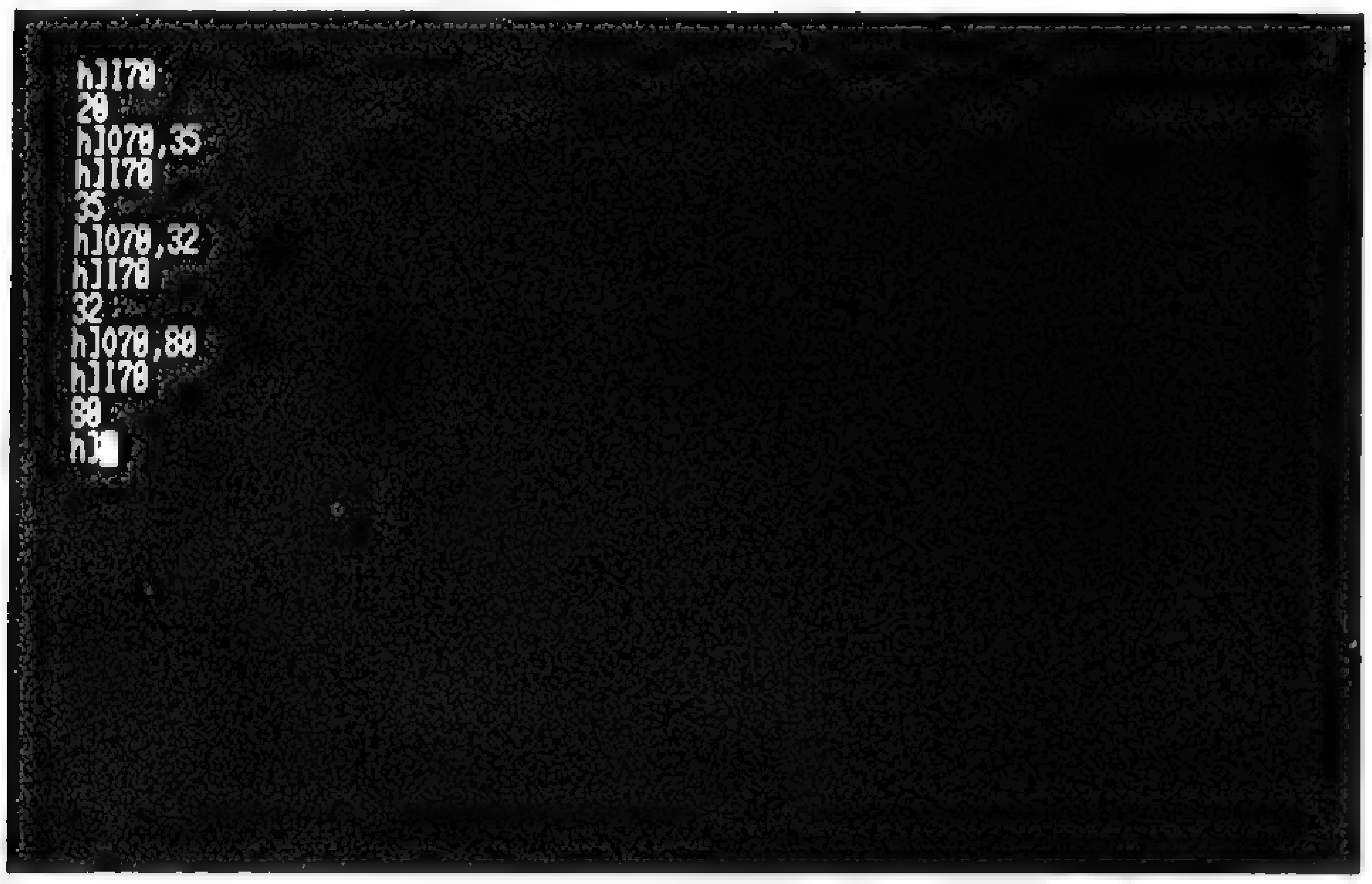
書式 G

ブレークポイントを設けずに，現在のプログラム・カウンタから実行します．

▶ I（インプット）

書式 Ipp　　pp：ポート・アドレス

入力ポートの値を読み，画面に表示します．

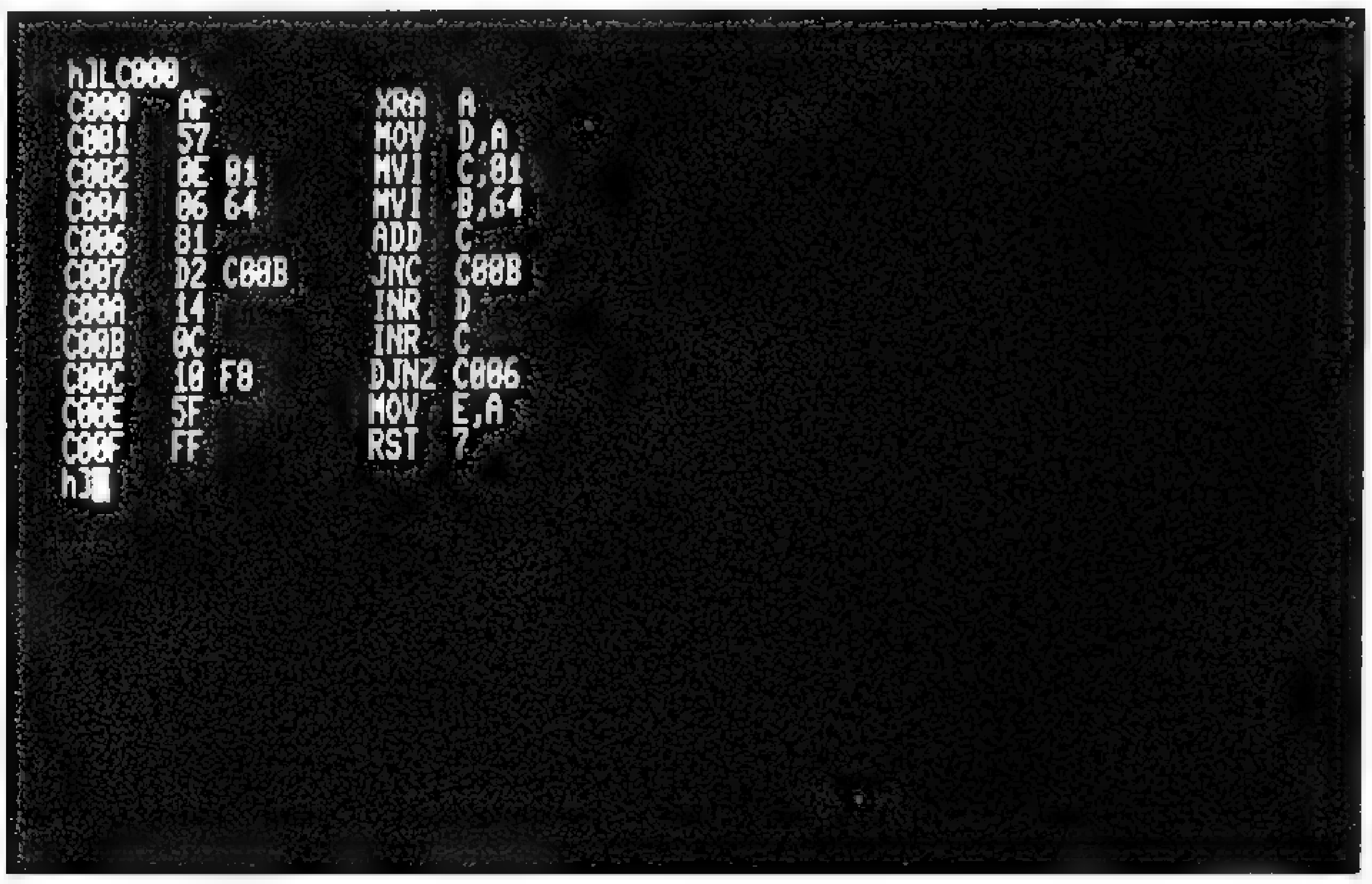


```
h]I70
20
h]O70,35
h]I70
35
h]O70,32
h]I70
32
h]O70,80
h]I70
80
h]
```

ポート70Hの内容を読み書き

▶ L（ディスアセンブル）

書式 Lssss, eeee



```
h]LC000
C000   AF           XRA  A
C001   57           MOV  D,A
C002   0E 01        MVI  C,01
C004   06 64        MVI  B,64
C006   81           ADD  C
C007   D2 C00B      JNC  C00B
C00A   14           INR  D
C00B   0C           INR  C
C00C   10 F8        DJNZ C006
C00E   5F           MOV  E,A
C00F   FF           RST  7
h]
```

C000Hから16バイトのディスアセンブル

ssssからeeeeまでのメモリの内容を逆アセンブルします.

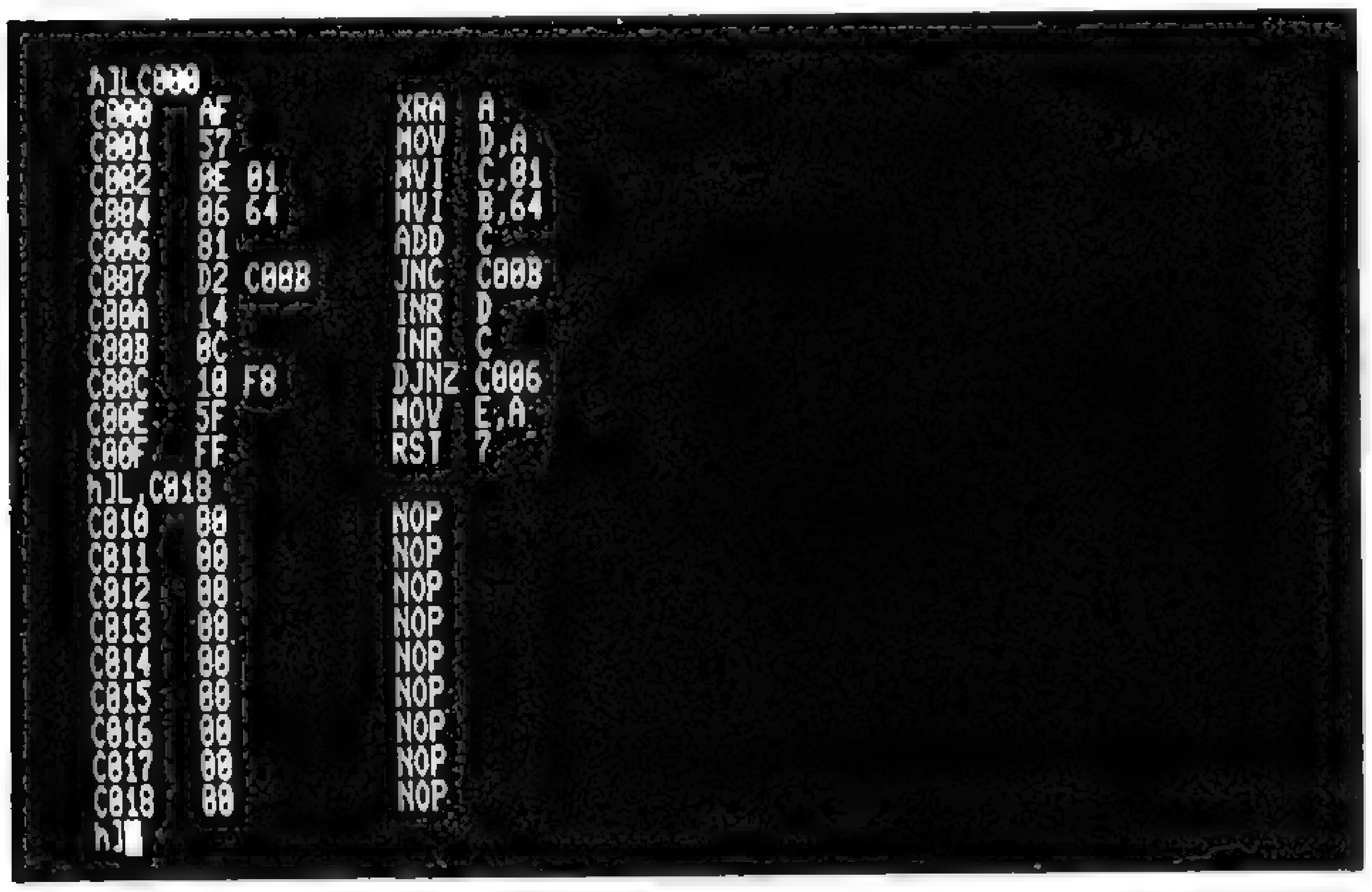
これによって表示されるニーモニックは，Ａコマンドのときと同じ8080用のインテル形です．従って，本書のリストとは一致しません.

書式 Lssss

ssssからssss＋15バイトのアドレスで指定されるメモリの内容を逆アセンブルします.

書式 L, eeee

前回のＬコマンドで表示したエンド・アドレスの直後のアドレスからeeeeで示されるアドレスの内容を逆アセンブルします.



```
h]LC000
C000    AF          XRA   A
C001    57          MOV   D,A
C002    0E 01       MVI   C,01
C004    06 64       MVI   B,64
C006    81          ADD   C
C007    D2 C00B     JNC   C00B
C00A    14          INR   D
C00B    0C          INR   C
C00C    10 F8       DJNZ  C006
C00E    5F          MOV   E,A
C00F    FF          RST   7
h]L,C018
C010    00          NOP
C011    00          NOP
C012    00          NOP
C013    00          NOP
C014    00          NOP
C015    00          NOP
C016    00          NOP
C017    00          NOP
C018    00          NOP
h]
```

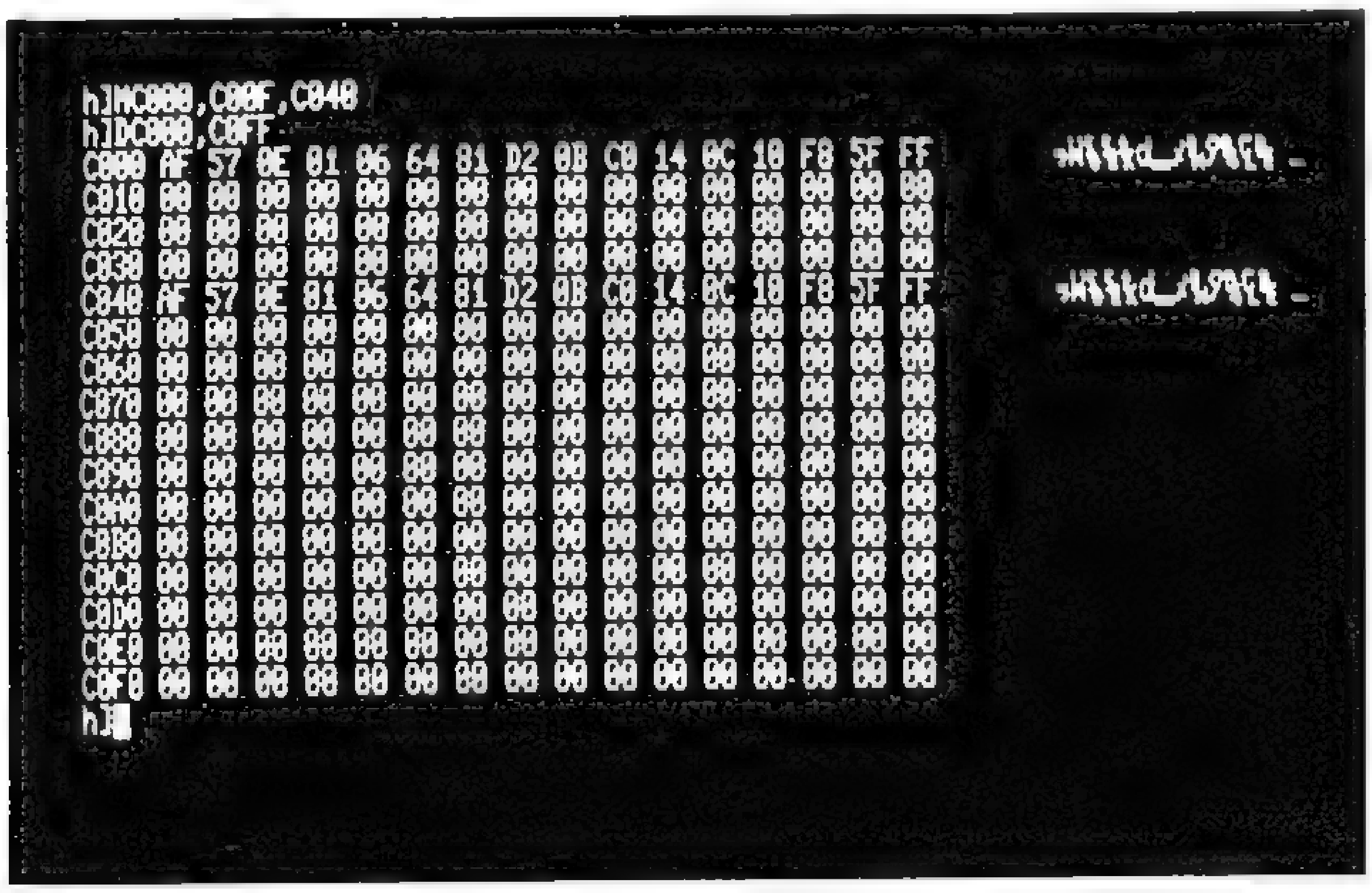
エンド・アドレスを指定した
ディスアセンブル

書式 L

前回のＬコマンドで表示した最終アドレスの直後のアドレスから16バイト分の内容を逆アセンブルします．なお，一度Ｌコマンドを実行した場合，リターンキーを押すだけで次のアドレスから16バイト分の逆アセンブルを行います.

▶ M（ムーブ・メモリ）

書式 Mssss, eeee, dddd　　　dddd：デスティネーション・アドレス



```
h]MC000,C00F,C040
h]DC000,C0FF
C000 AF 57 0E 01 06 64 81 D2 0B C0 14 0C 10 F8 5F FF
C010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C040 AF 57 0E 01 06 64 81 D2 0B C0 14 0C 10 F8 5F FF
C050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
h]
```

C000HからC00FHまでのメモリの内容をC040Hからブロック転送

アドレスssssからeeeeで示されるメモリの内容を，アドレスddddから始まるメモリへ転送します.

▶O（アウトプット）

書式 Opp, dd　　dd：データ

出力ポート・アドレスppで示されるポートへ，データddを出力します.

▶P（プリントスイッチ）

このコマンド以後に実行される，Dコマンド，Lコマンド，CTRL-Dコマンドの内容を，プリンタに画面と同時に出力します. この場合プロンプトはh)となります. もとに戻すには，もう一度Pコマンドを実行します.

▶R（リード・テープ）

書式 Rnnnnnn　　nnnnnn：ファイル・ネーム

nnnnnnで示されるファイル・ネームのマシン語プログラムをカセットテープからロードします.

書式 R1：nnnnnn

1200ボーでカセットテープからロードします（1：が指定されない場合も同じ働きをする）.

書式 R2：nnnnnn

600ボーでカセットテープからロードします.

▶S（セット・メモリ）

書式 Sssss

アドレスssssで示されるメモリに書き込みを行います. Eコマンドと異なりスクリーンエディットはできません. 次のアドレスに移るには，スペースバーを押し，前のアドレスに戻るには CTRL － B を押します.

▶TM（テスト・メモリ）

PC-8801の全RAMをテストします. またこのコマンドを実行するとすべてのメモリが書き換わってしまいますから注意が必要です.

▶V（ベリファイ・テープ）

書式 Vnnnnnn

nnnnnnで示されるファイル・ネームのマシン語プログラムとメモリの内容を比較します.

▶ W（ライト・テープ）

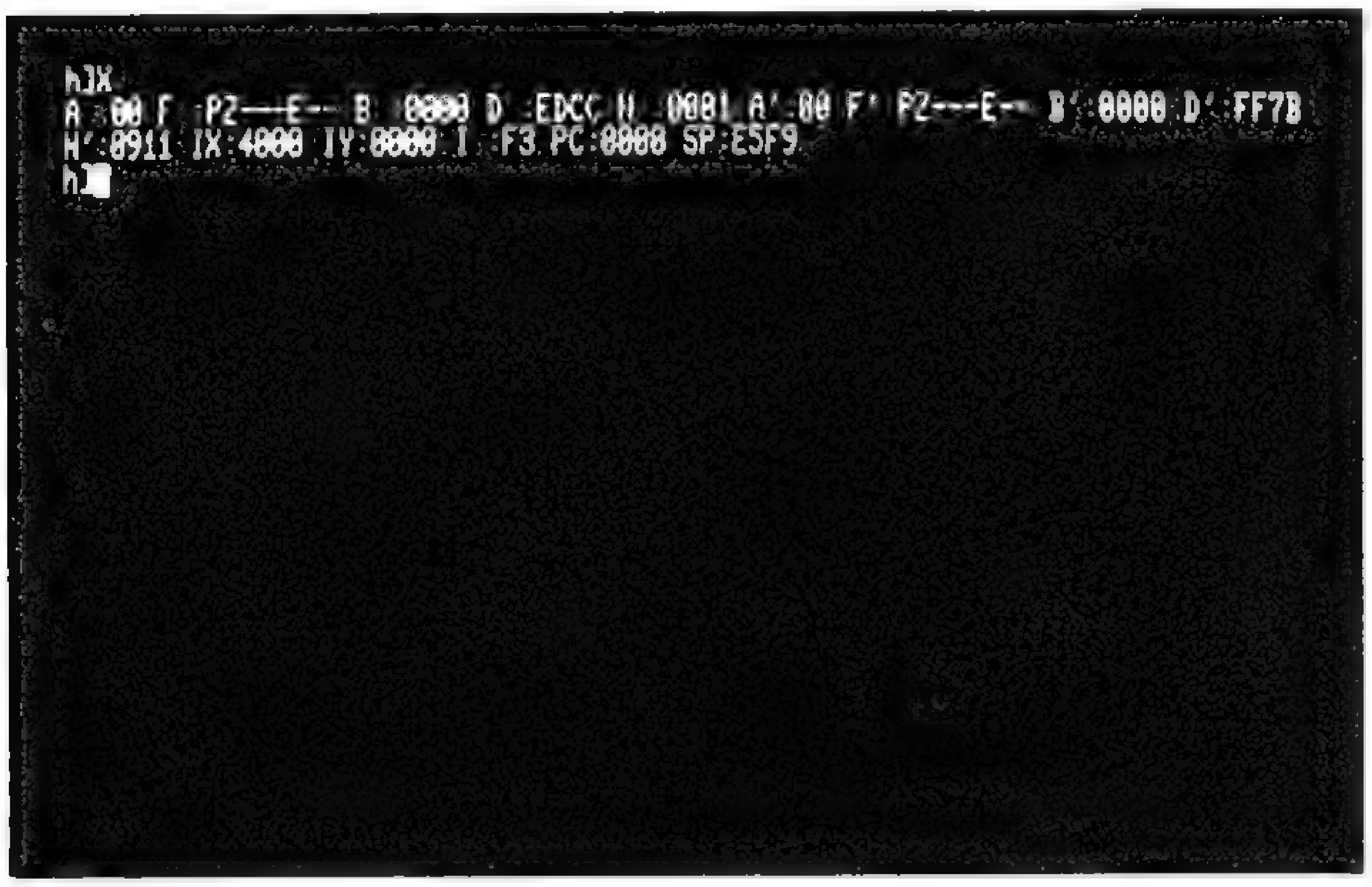
書式 Wnnnnnn, ssss, eeee

アドレスssssからeeeeで示されるメモリの内容をファイル・ネームnnnnnnでカセットテープにセーブします．ボーレートについては，Rコマンドに準じます．

▶ X（イグザミン・レジスタ）

書式 X

現在のCPUが保持している各レジスタの内容を表示します．

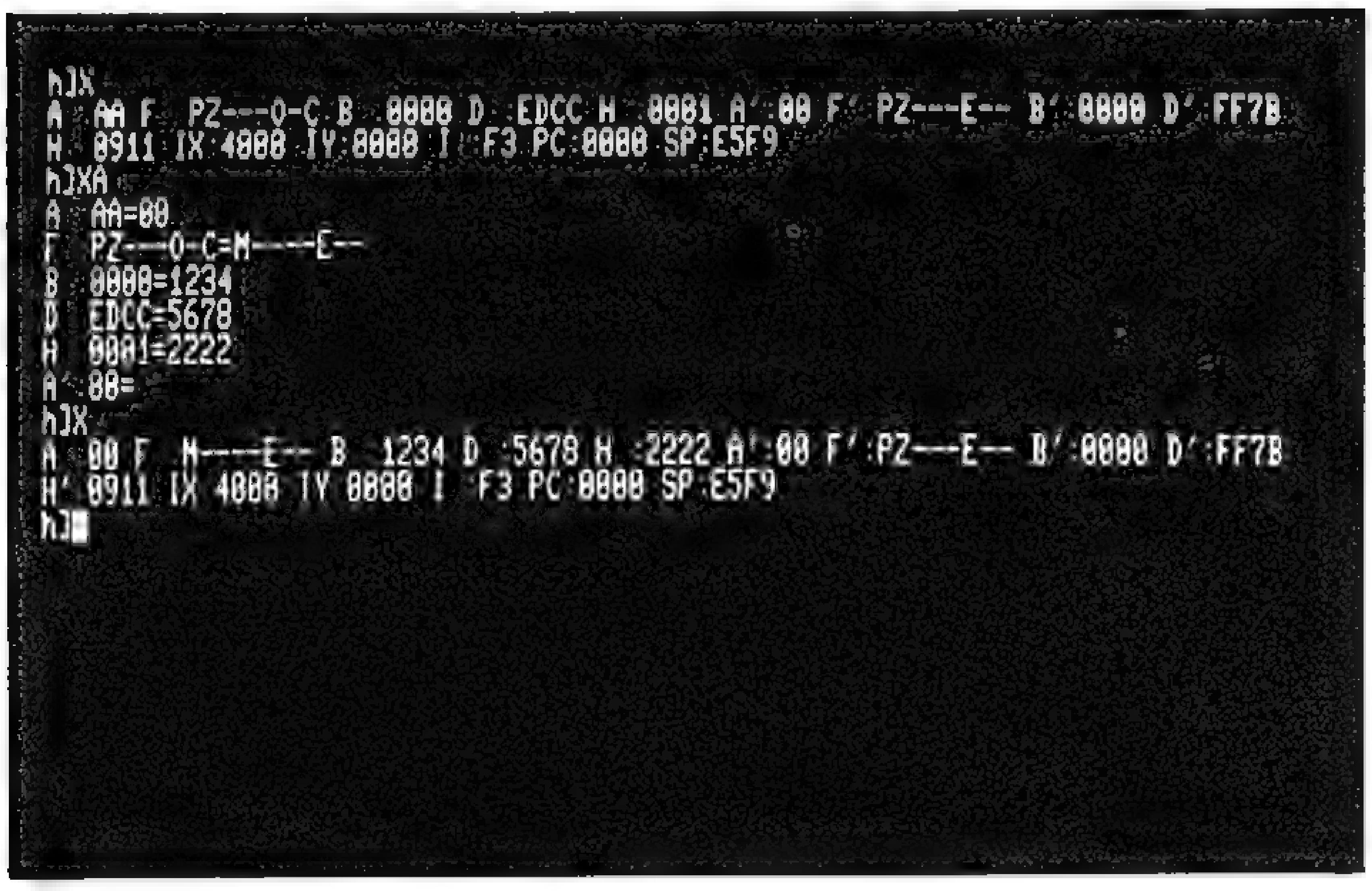


```
h]X
A :00 F :PZ---E-- B :0000 D :EDCC H :0001 A':00 F':PZ---E-- B':0000 D':FF7B
H':0911 IX:4000 IY:0000 I :F3 PC:0000 SP:E5F9
h]█
```

レジスタの内容の表示

書式 Xr

r で示されるレジスタの内容の変更を行います．



```
h]X
A :AA F :PZ---O-C B :0000 D :EDCC H :0001 A':00 F':PZ---E-- B':0000 D':FF7B
H :0911 IX:4000 IY:0000 I :F3 PC:0000 SP:E5F9
h]XA
A :AA=00
F :PZ---O-C=M----E--
B :0000=1234
D :EDCC=5678
H :0001=2222
A':00=
h]X
A :00 F :M----E-- B :1234 D :5678 H :2222 A':00 F':PZ---E-- B':0000 D':FF7B
H':0911 IX:4000 IY:0000 I :F3 PC:0000 SP:E5F9
h]█
```

A, F, B, D, Hの内容変更

▶ CTRL - B

ベーシック・モードに戻ります．

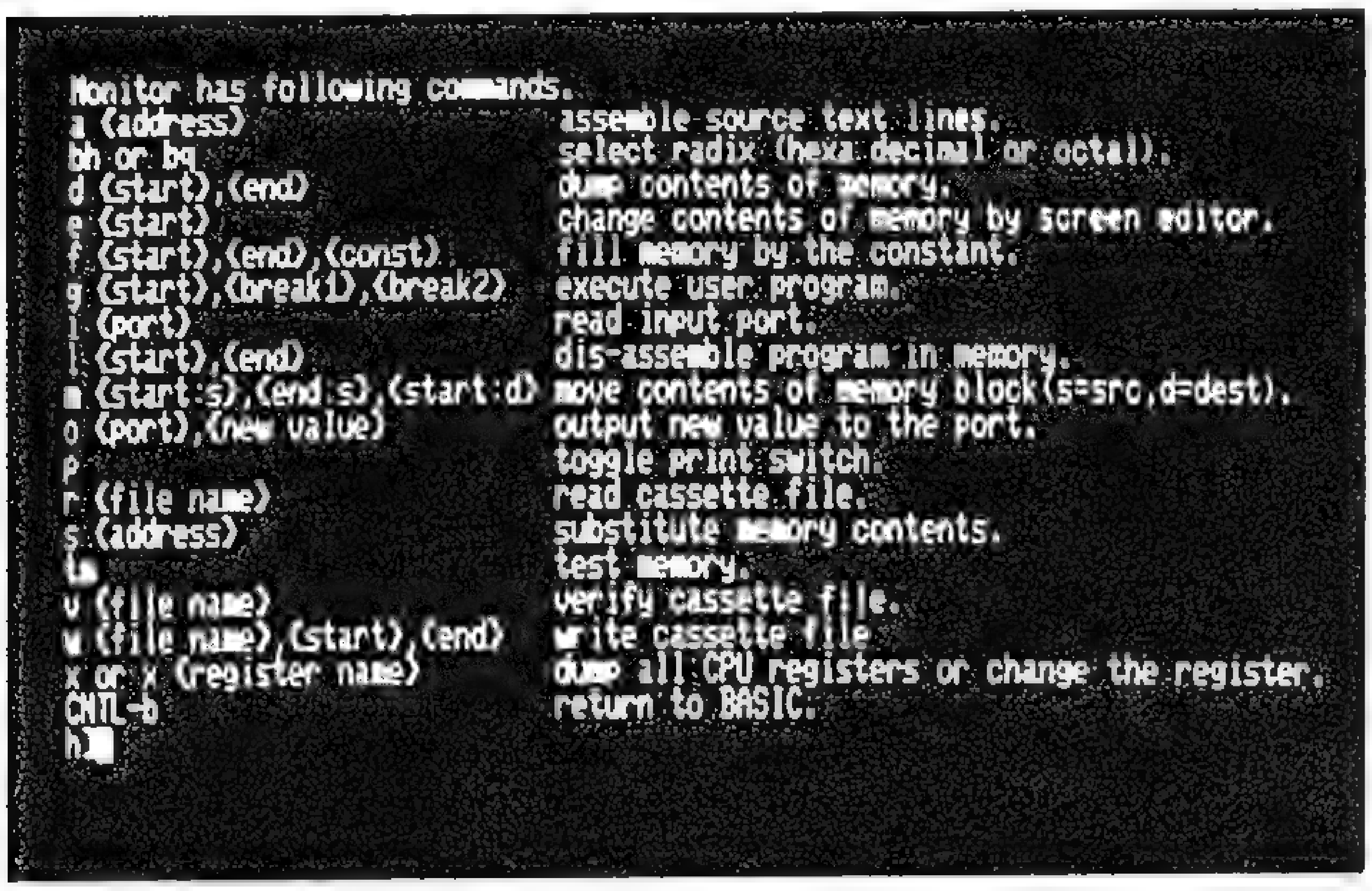
▶ CTRL - Aまたは HELP

CTRL - Aには次の3種類があります．

モニタのプロンプトが表示された入力待ちのときに，CTRL-AまたはHELPを入力すると，モニタで使用できるコマンドの種類とその書式と機能が表示されます．

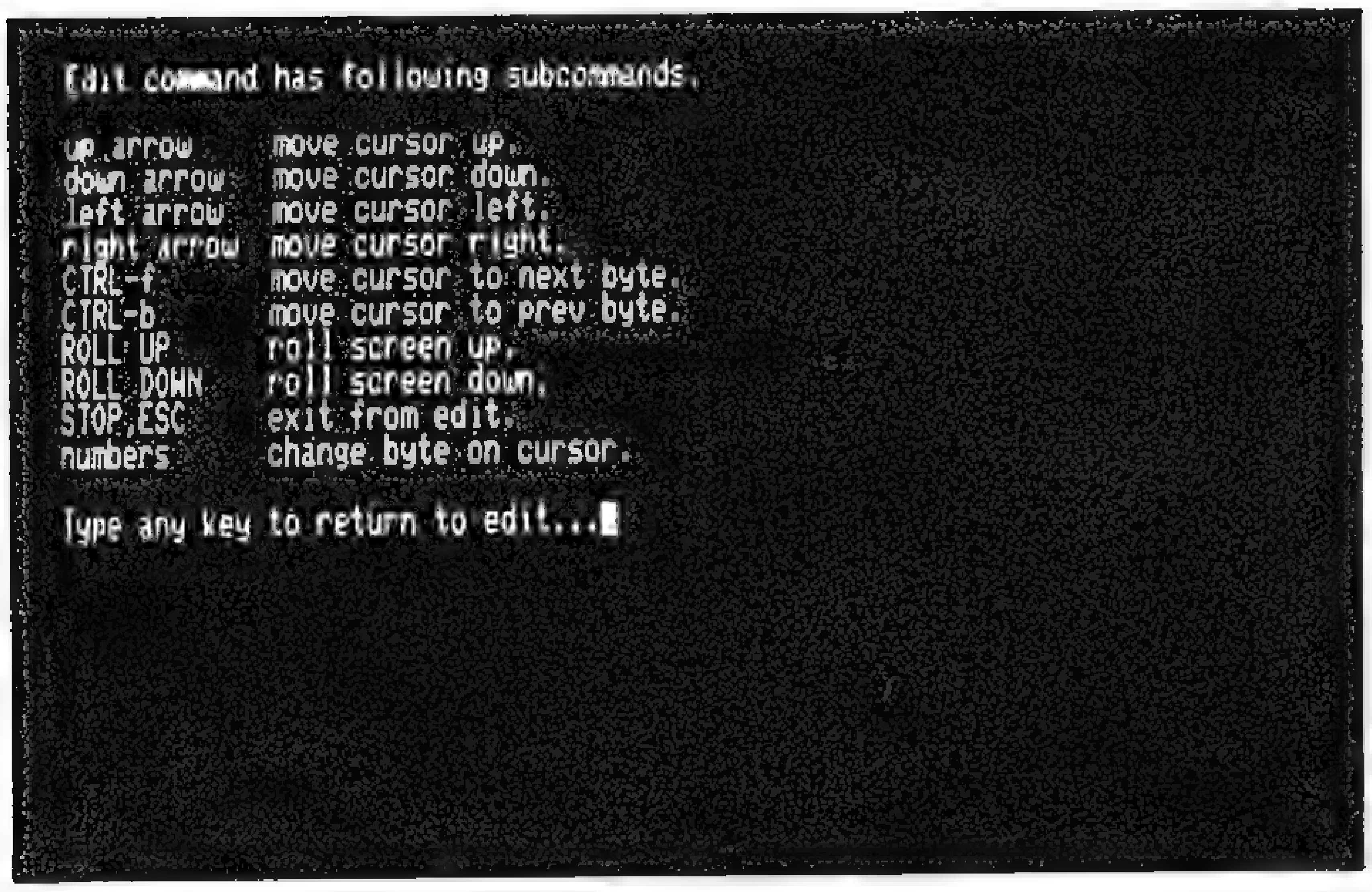
Eコマンドでエデット・モードに入ったときに，CTRL-AまたはHELPを入力すると，エデット・モードで使えるコマンドを表示します．

Xrコマンドで，フラグ･レジスタ（FまたはF'）の入力待ちのときに，CTRL-AまたはHELPを入力すると，フラグの順序とその入力の仕方が表示されます．



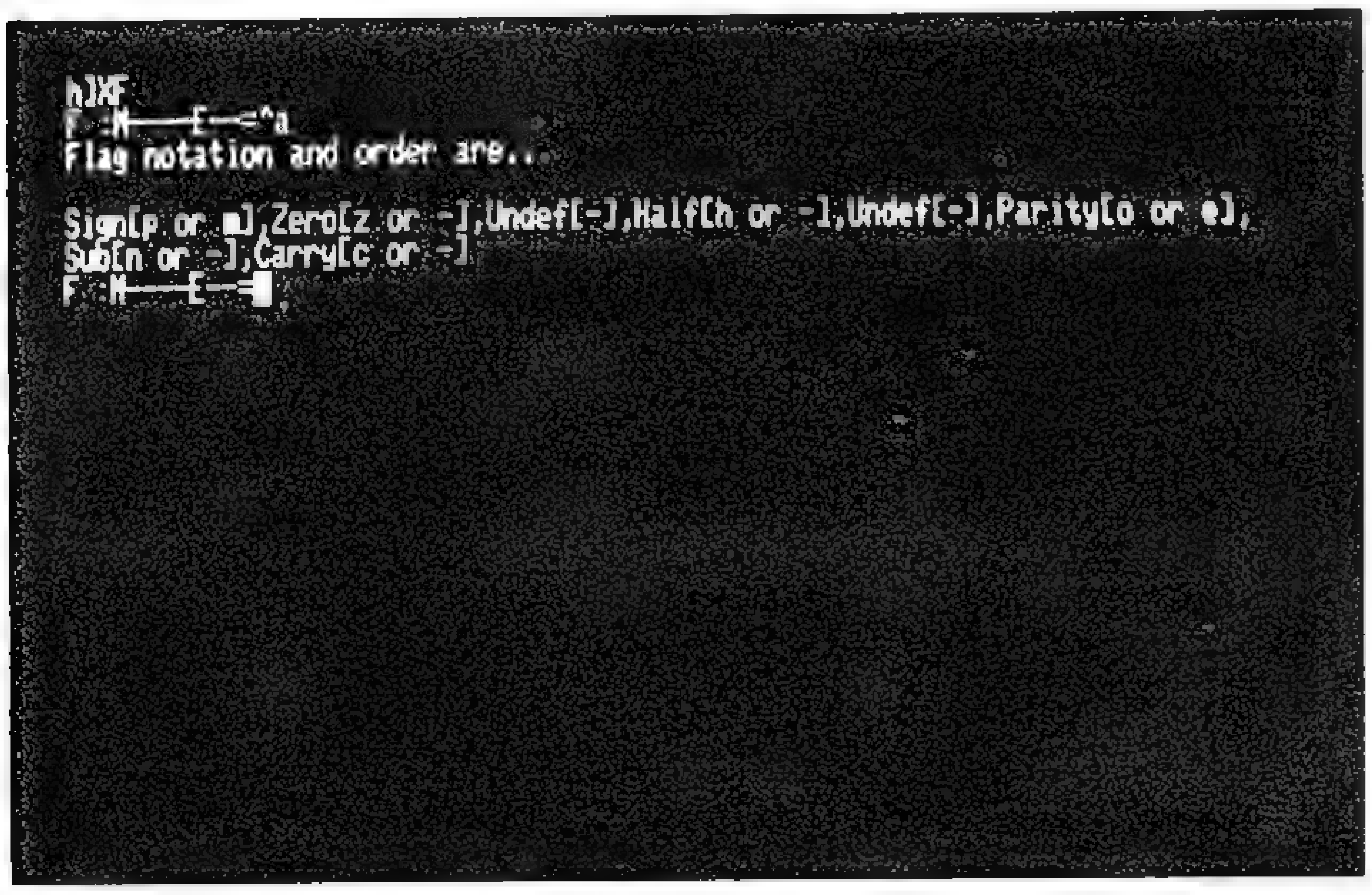
```
Monitor has following commands.
a (address)                      assemble source text lines.
bh or bq                         select radix (hexa decimal or octal).
d (start),(end)                  dump contents of memory.
e (start)                        change contents of memory by screen editor.
f (start),(end),(const)          fill memory by the constant.
g (start),(break1),(break2)      execute user program.
i (port)                         read input port.
l (start),(end)                  dis-assemble program in memory.
m (start:s),(end:s),(start:d)    move contents of memory block(s=src,d=dest).
o (port),(new value)             output new value to the port.
p                                toggle print switch.
r (file name)                    read cassette file.
s (address)                      substitute memory contents.
tm                               test memory.
v (file name)                    verify cassette file.
w (file name),(start),(end)      write cassette file
x or x (register name)           dump all CPU registers or change the register.
CNTL-b                           return to BASIC.
h]█
```

モニタのコマンド・レベルの表示



```
Edit command has following subcommands.

up arrow      move cursor up.
down arrow    move cursor down.
left arrow    move cursor left.
right arrow   move cursor right.
CTRL-f        move cursor to next byte.
CTRL-b        move cursor to prev byte.
ROLL UP       roll screen up.
ROLL DOWN     roll screen down.
STOP,ESC      exit from edit.
numbers       change byte on cursor.

Type any key to return to edit...█
```

エディット・モードの表示



```
h]XF
F :M———E——=^a
Flag notation and order are...

Sign[p or m],Zero[z or -],Undef[-],Half[h or -],Undef[-],Parity[o or e],
Sub[n or -],Carry[c or -]
F :M———E——=█
```

Fレジスタの入力待ちのときの表示

以上のコマンドは標準でサポートされているものですが，以下のコマンドは，DISK-BASICでのみサポートされます．なお，５インチ片面ドライブを使用しているときは，サーフェス番号を指定してはいけません．

▶CTRL-D

書式 CTRL-Ddr, sur, tr 1 , sec 1 , tr 2 , sec 2

ドライブ番号dr，サーフェス番号sur，トラックtr 1，セクタsec 1から，トラックtr 2，セクタsec 2 のディスケット上の内容を表示します．

書式 CTRL-Ddr, sur, tr, sec

ドライブ番号dr，サーフェス番号sur，トラックtr，セクタsecで示される１セクタのディスケットの内容を表示します．

▶CTRL-R

書式 CTRL-Rdr, sur, tr, sec, ssss, eeee

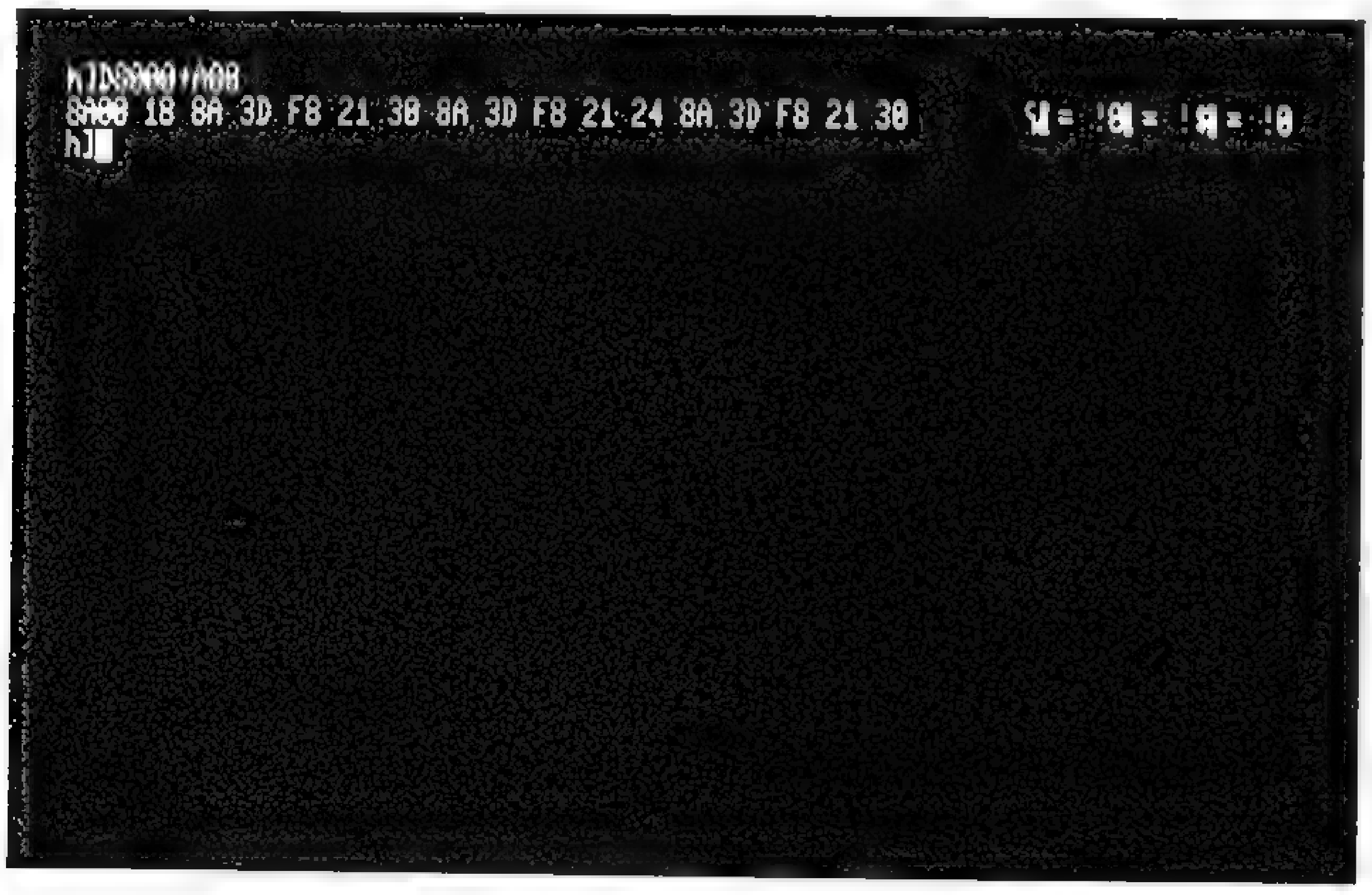
ドライブ番号dr，サーフェス番号sur，トラックtr，セクタsecから始まるディスケットの内容を，アドレスssssからeeeeまでのメモリに書き込みます．

▶CTRL-W

書式 CTRL-Wdr, sur, tr, sec, ssss, eeee

ドライブ番号dr，サーフェス番号sur，トラックtr，セクタsecから始まるディスケット上のセクタよりアドレスssssからeeeeで示されるメモリ上の内容を書き込みます．

■ 計算式の利用



```
h]D8000+A00
8A00 18 8A 3D F8 21 30 8A 3D F8 21 24 8A 3D F8 21 30    [illegible]
h]
```

計算式によるスタート・アドレスの指定

数値を入力する際には，計算式(加／減算のみ)が利用できます．例えば，8000HにA00Hを足したアドレスを表示するには次のように入力します．

```
h] D8000＋A00 ↲
```

8A00H番地を先頭に16バイト表示されました．

■ 誤り訂正

モニタ・モードにおいてコマンド入力時には INS DEL ，カーソル・コントロールキーも働きませんが，数値を誤って入力してしまった場合(もちろんリターンキーを押す前の状態)にはどのように訂正すればよいのでしょうか．この場合は続いて正しい数値を入力します．例えば，Dコマンドで次のように入力すると最後の4桁が有効になり，8000H番地から16バイト表示されます．

```
                     有効
h] D ○○○○○‥ 8000
     誤って入力した部分
```

4.3　メモリマップとメモリの使用法

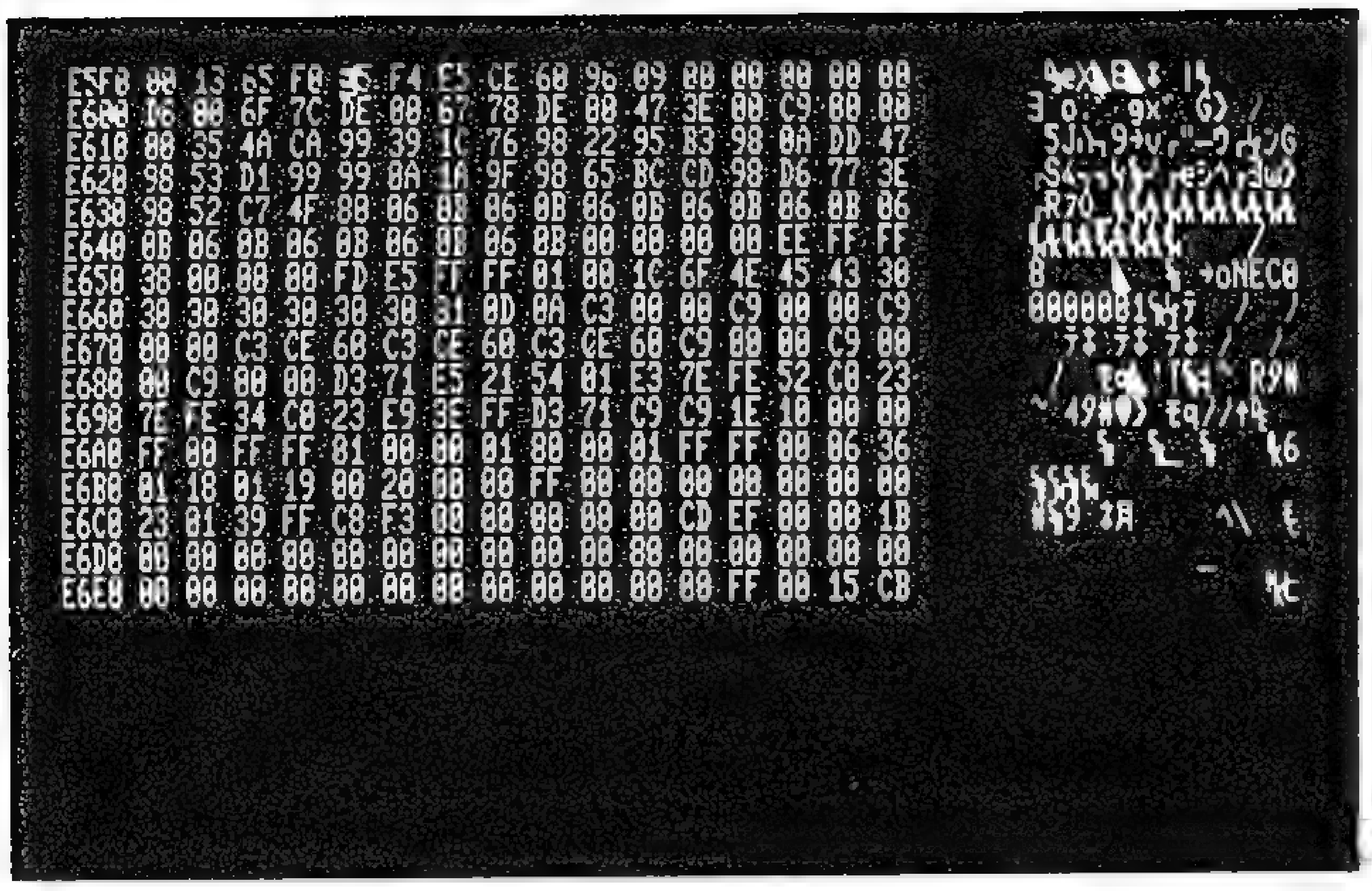
さて，マシン語をBASICとリンクして使う際に必ず問題となるのが，作成したプログラムやデータをメモリ上のどこに置くかということです．BASICプログラムの場合，プログラムの格納領域(テキスト領域)とデータの格納領域(変数エリア)はインタプリタが管理してくれますから特別なことを考える必要はなく，行番号，ステートメントそしてリターンキーの入力によってプログラムがメモリに記憶されます．ところが，マシン語プログラムの場合，モニタやアセンブラを使用してプログラムをメモリ上に格納しなければなりません．その場合，メモリ・アドレスをいい加減に決めると，プログラム自身は正常であっても実行した場合に暴走することがあります．

BASICだけを利用している限りめったに暴走という状態は起こりません．しかし，ユーザの作成したマシン語プログラムは，格納するアドレスをユーザ自身が管理しなければなりません．

4.3.1　危険地帯

本体の電源ON直後(またはリセット直後)モニタ・モードに入ります．モニタのEコマンドを使用して，E5F4Hから256バイトを表示させ，次のように入力します．

```
h] EE5F4 ↲
```



```
E5F0 00 13 65 F0 E5 F4 E5 CE 60 96 09 00 00 00 00 00
E600 D6 00 6F 7C DE 00 67 78 DE 00 47 3E 00 C9 00 00
E610 00 35 4A CA 99 39 1C 76 98 22 95 B3 98 0A DD 47
E620 98 53 D1 99 99 0A 1A 9F 98 65 BC CD 98 D6 77 3E
E630 98 52 C7 4F 00 06 0B 06 0B 06 0B 06 0B 06 0B 06
E640 0B 06 0B 06 0B 06 0B 06 0B 00 00 00 00 EE FF FF
E650 38 00 00 00 FD E5 FF FF 01 00 1C 6F 4E 45 43 30
E660 30 30 30 30 30 30 31 0D 0A C3 00 00 C9 00 00 C9
E670 00 00 C3 CE 60 C3 CE 60 C3 CE 60 C9 00 00 C9 00
E680 00 C9 00 00 D3 71 E5 21 54 01 E3 7E FE 52 C0 23
E690 7E FE 34 C0 23 E9 3E FF D3 71 C9 C9 1E 10 00 00
E6A0 FF 00 FF FF 81 00 00 01 80 00 01 FF FF 00 06 36
E6B0 01 18 01 19 00 20 00 00 FF 00 00 00 00 00 00 00
E6C0 23 01 39 FF C8 F3 00 00 00 00 00 CD EF 00 00 1B
E6D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E6E0 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 15 CB
```

E5F4H番地を書き換えると!?

E5F4H番地でカーソルが点滅して，その値はE５になっていると思います．これをE０に書き換えると，どうです，リセットされて，N-BASICが動き出すか，暴走したのではないでしょうか．このE5F4H番地は，実はスタック領域の一部だったのです．このように，マシン語プログラムはメモリ上のどこへでも置けるものではありません．では，どこを選べばよいのでしょう．

4.3.2　メモリマップ

図4.1に，PC-8801のメモリマップを示します．メモリマップとは，メモリの用途別使用割り当てのことです．μPD780はアドレス・バスが16ビットですから，$2^{16}=65536$バイト(64Kバイト)のメモリしか扱えないはずなのですが，実際には標準でRAMが112Kバイト，ROMが72Kバイトの合計184Kバイトももっています．これらのメモリのうち，CPUが同時に扱えるのは64Kバイトで，N_{88}-BASICはこれらのメモリを巧みに切り換えて使用します．

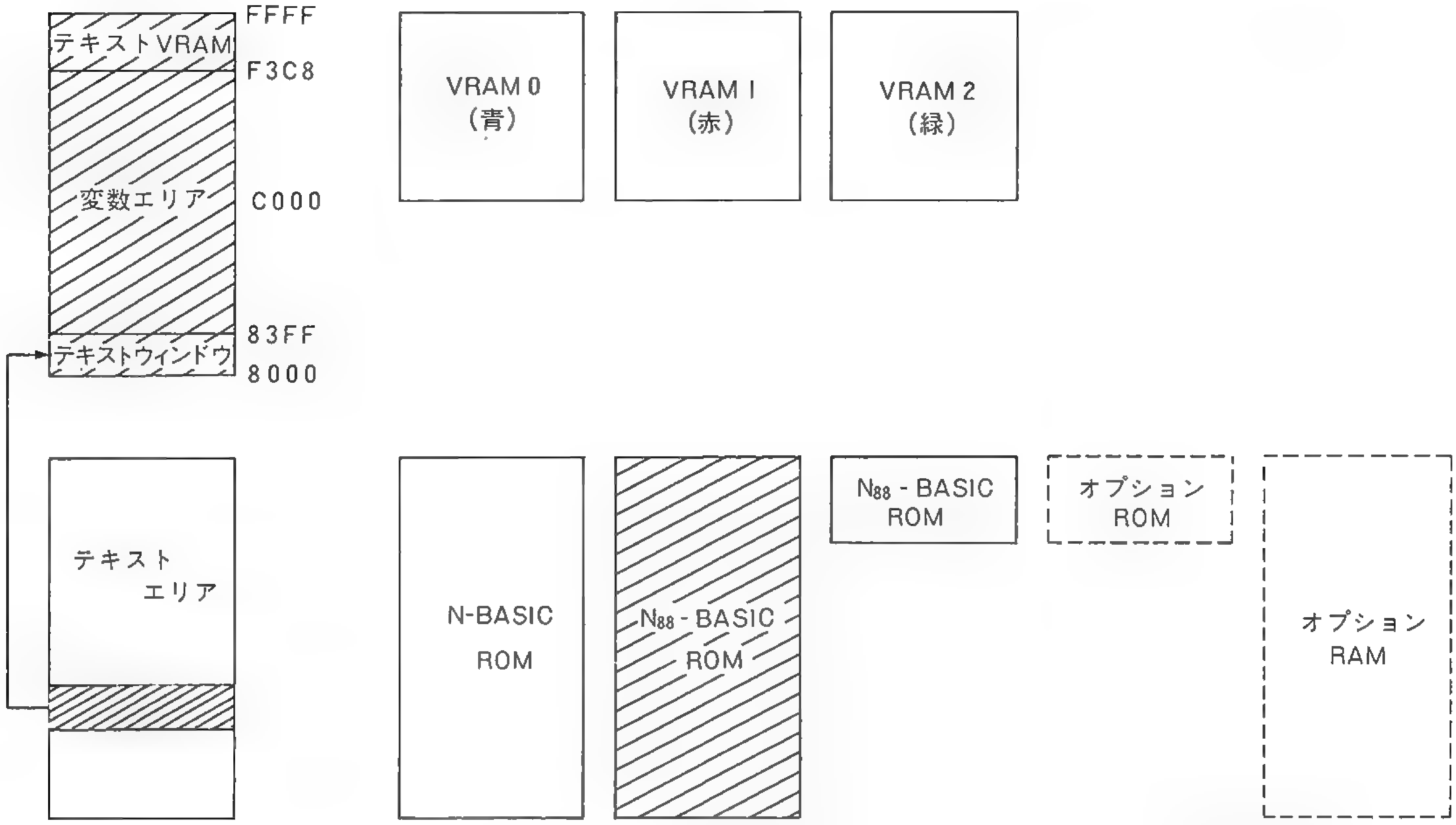
N_{88}-BASICモードでの起動時(普段直接CPUに接続されている部分)のメモリマップは，図４．２の実線で示したようになっています．この中には大別して，(1)N_{88}-BASIC ROMエリア，(2)テキスト・ウィンドウ，(3)ファイル・バッファ，(4)変数エリア，(5)BASICワーク・エリア，(6)テキストVRAM(単にVRAMともいう)エリアがあります．

さて，以上の中からマシン語領域に使用できるメモリ領域を考えてみます．

(1)のN_{88}-BASIC ROMエリアは，ROMですから書き込むことはできません．書き込めないというのは，とりもなおさずマシン語プログラムを置けないということです．

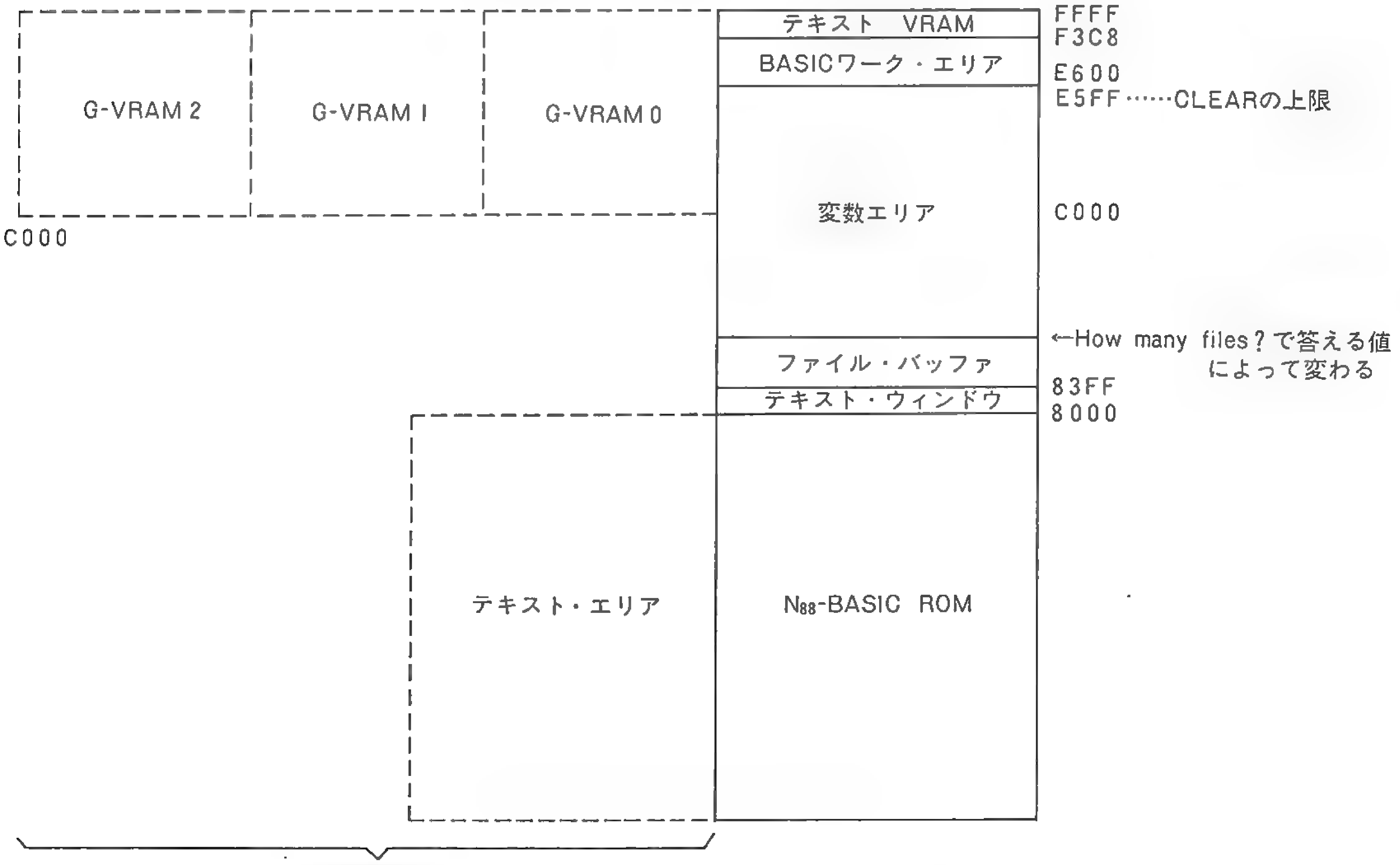
(2)のテキスト・ウィンドウについては，多少説明が必要です．これは非常に特殊なエリアで，N_{88}-BASICではテキスト・エリアは，図4.2の通りROMエリアの裏(アドレスの重複した位置)にありますから，こちらを選択するとBASIC-ROMが切り離されてしまい，BASICのプログラム続行が不可能となります．そのため，ハードウェアで，テキスト・ウィンドウと呼ばれるものを設けてあります.このテキスト・ウィンドウは，I／Oポートへのコマンドによって BASICテキスト・エリアの１Kバイトがみえるようにしてあるエリアです．つまりN_{88}-BASICインタプリタは，このテキスト・ウィンドウを通じてBASICプログラムを読み，これを実行するのです．従って，この部分もマシン語プログラムエリアとしては使用できません(この領域の使用は，全く不可能というのでは

ない．『PC-8801 グラフィックスのすべて』に掲載されているGlibではこのエリアを使用している)．


テキストVRAM
FFFF
F3C8
変数エリア
C000
83FF
テキストウィンドウ
8000
VRAM 0
(青)
VRAM 1
(赤)
VRAM 2
(緑)
テキスト
エリア
N-BASIC
ROM
N88-BASIC
ROM
N88-BASIC
ROM
オプション
ROM
オプション
RAM

＊：N88-BASICモードでは，テキスト・エリアの１Kバイトが変数エリアのテキストウィンドウに転送されて処理される

図4.1　PC-8801のメモリマップ(1)


G-VRAM 2
G-VRAM 1
G-VRAM 0
C000
テキスト　VRAM
BASICワーク・エリア
変数エリア
ファイル・バッファ
テキスト・ウィンドウ
テキスト・エリア
N88-BASIC　ROM
FFFF
F3C8
E600
E5FF……CLEARの上限
C000
←How many files？で答える値によって変わる
83FF
8000

これらは直接操作できない

図4.2　PC-8801のメモリマップ(2)

(3)のファイル・バッファとは，N$_{88}$-BASICがファイルとの入出力のやりとりに必要な情報を記憶するための領域です．この領域は，“How many files ?”に対して入力した数によって大きさが変化します．ここもマシン語領域には使用できません．

(4)の変数エリアは，BASICが変数の値を記憶しておくエリアですのでやはりマシン語には使用できません．

(5)のワーク・エリアは，BASICが動作中に必要な情報を記憶するエリアです．やはり，マシン語には使用できません．

(6)のテキストVRAMエリアは，ここのメモリの内容がそのままキャラクタ画面に表示されます．ということは，画面に表示される内容が変わるたびにメモリの内容が書き換えられてしまいますので，やはりマシン語には使用できません．

4.3.3　マシン語領域の確保

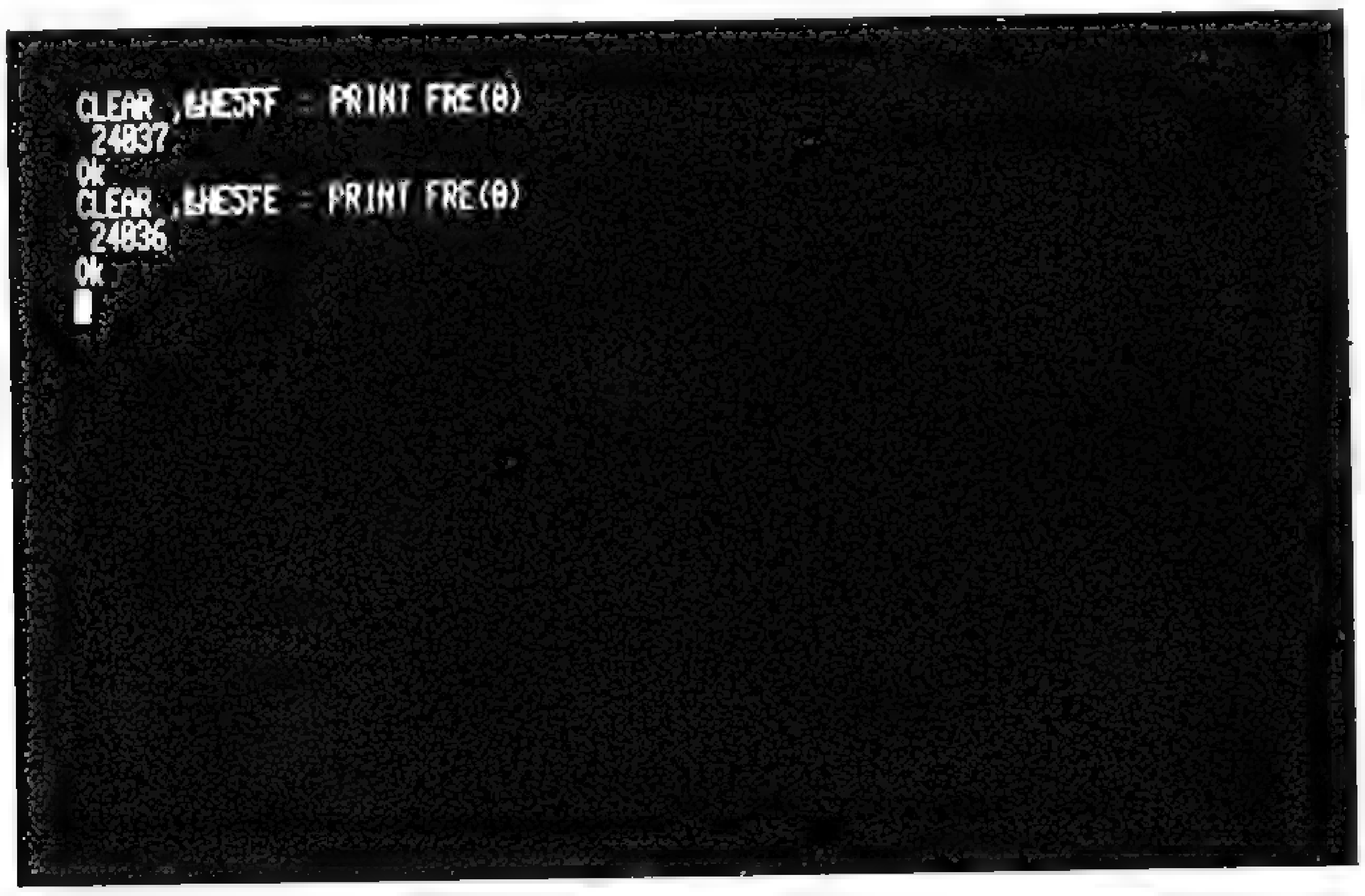
さて，(1)～(6)の各エリアはマシン語領域としては不適当でしたが，マシン語領域を確保するには，ユーザがBASICのCLEARというコマンドを使ってマシン語のためのエリアを確保しなければなりません．書式は次のようになります．

CLEAR〔〈文字変数領域の大きさ〉〔,〈メモリの上限アドレス〉〔,〈スタック領域の大きさ〉〕〕〕
第1パラメータ　　第2パラメータ　　第3パラメータ

N$_{88}$-BASICにおいて，第1パラメータは意味をもちませんし，また第3パラメータは，普通に使用する分にはほとんど問題とならないので設定する必要はありません(デフォルトでは512バイト)．ここで問題となるのは第2パラメータです．この第2パラメータの数値の意味が，BASICで使用する変数エリアの上限を決定します．例えば，

```
CLEAR ,&HBFFF
```

とすると，BASICで使用するメモリの上限をBFFFH番地に決定し，C000H番地以降は使用しないことを宣言します．次のプログラムをダイレクトステートメントで実行してみてください．



```
CLEAR ,&HE5FF : PRINT FRE(0)
 24837
Ok
CLEAR ,&HE5FE : PRINT FRE(0)
 24836
Ok
█
```

FRE(0)の値

```
CLEAR ,&HE5FF：PRINT FRE(0)
CLEAR ,&HE5FE：PRINT FRE(0)
```

第２パラメータを１バイト変えることにより，FRE(0)の値つまり変数エリアが１バイト減ったことがわかります．これを次々と繰り返すと，FRE(0)の値が１バイトずつ減っていきます．こうして，BASICの使用する変数エリアの上限は次々と下がりますが，ワーク・エリアの始まるメモリのアドレスは，相変らずE600H番地からです．つまり，BASICの変数エリアと，BASICワーク・エリアとの間にすき間ができたことになり，このすき間こそが，ユーザがマシン語領域として自由に使用できるメモリ領域となるのです．先程のダイレクトステートメントのプログラムでは，最初の例のマシン語領域は０バイト，２つ目の例ではE5FFH番地の１バイトのみが確保されたことになりますが，１バイトのみではマシン語プログラムとして動作させることはできませんので，必要に応じたマシン語領域を確保しなければなりません．例えば，１Kバイトのマシン語領域を確保したいならば，次のように実行します．

```
CLEAR ,&HE1FF
```

これでマシン語領域としてE200H番地からE５FFH番地までが確保できたことになります．

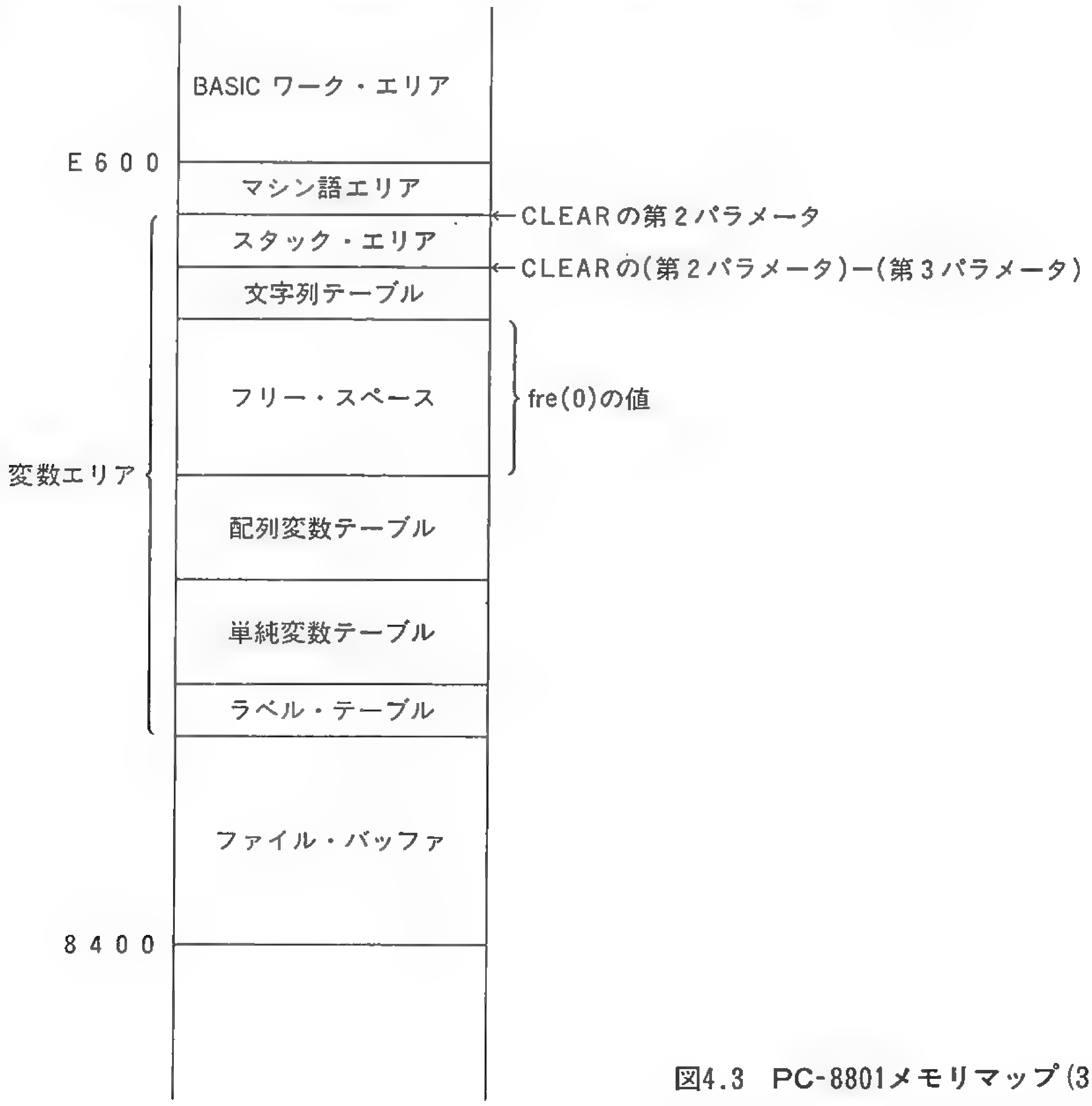

BASIC ワーク・エリア
E600
マシン語エリア
←CLEARの第２パラメータ
スタック・エリア
←CLEARの(第２パラメータ)－(第３パラメータ)
文字列テーブル
フリー・スペース
fre(0)の値
変数エリア
配列変数テーブル
単純変数テーブル
ラベル・テーブル
ファイル・バッファ
8400

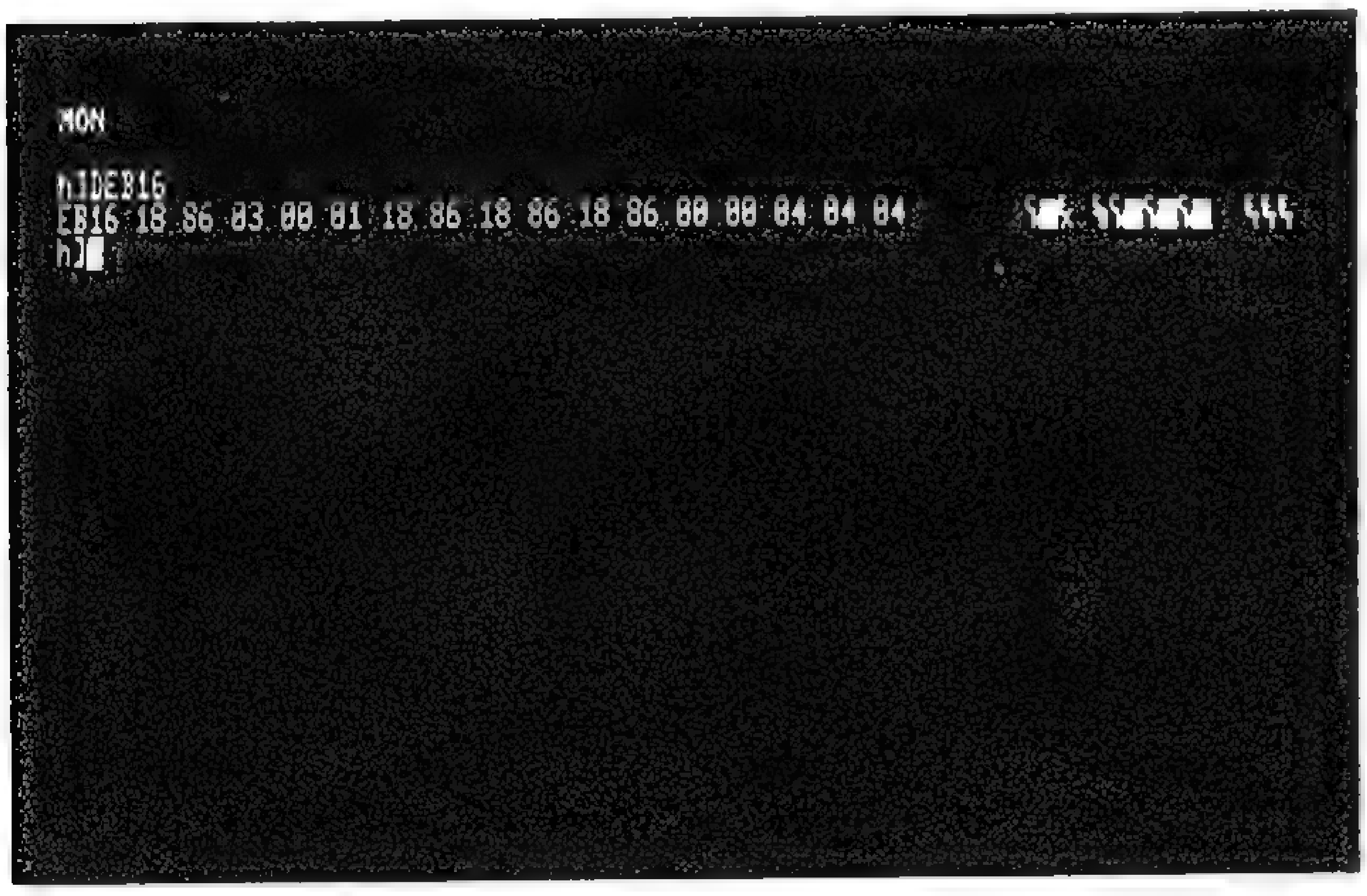
図4.3　PC-8801メモリマップ(3)

4.3.4 MAXエリア

ところで，CLEAR文の第２パラメータの値はどこまで許されるのでしょうか．つまり，マシン語領域は一体どれくらい確保できるかということです．ここで先程変数エリアとして説明した部分のメモリマップを図4.3に示します．この図の中で，ラベル・テーブルはBASICプログラム中で使用したラベルを，変数テーブルは使用した変数の値を記憶します．従って，これらの使用するメモリ領域はプログラムにより変化しますので，マシン語領域を大量にとりたい場合は少々困ったことになります．つまり，できるだけ大きくマシン語領域を確保するためにCLEARの第２パラメータの値をかなり小さめにとった場合，BASICプログラムにおいて変数およびラベルを大量に使用すると，“Out of memory (OM Error)”，または“Out of string space (OS Error)”となってしまいます．もし，BASICプログラムにおいてラベルおよび変数などを使用しなければ，CLEARの第２パラメータは，ラベル・テーブルのスタート・アドレス＋$533_{(10)}$まで許されるようです．

さて，ラベル・テーブルのスタート・アドレスはどのようにして調べればよいのかといいますと，この値は，“How many files？”に答える値によって，またROM-BASICとDISK-BASICによっても大きく変わります．この値は，BASICワーク・エリアのEB16H，EB17H番地に入っています．それではこの値を調べてみましょう．まずモニタ・モードに入って，Dコマンドを実行します．

```
h]DEB16 ↵
```

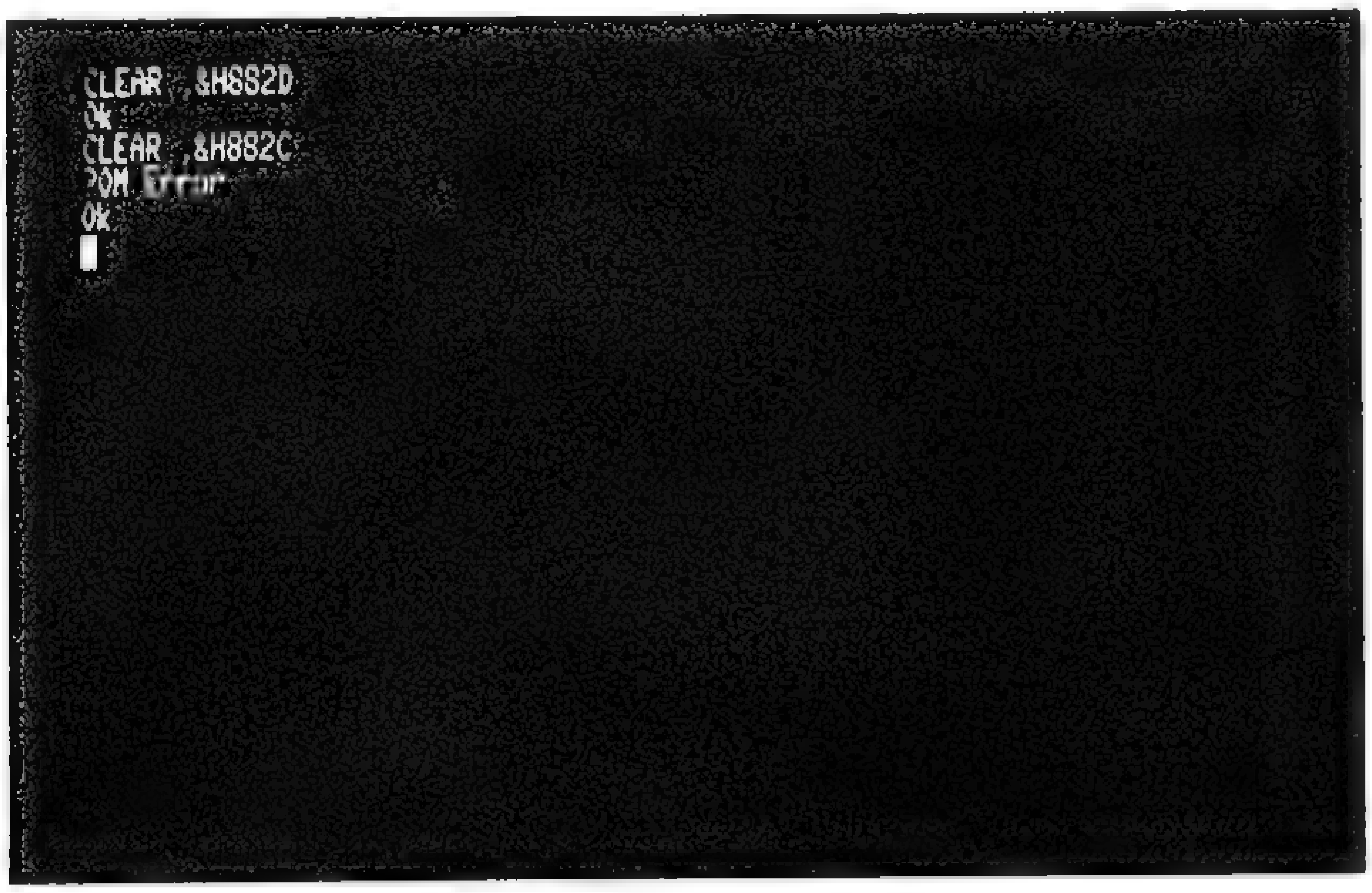


```
MON

h]DEB16
EB16 18 86 03 00 01 18 86 18 86 18 86 00 00 04 04 04
h]
```

ラベル・テーブルのスタート・アドレス

この写真の例では，ROM-BASICモードの，“How many files（0-15）？”に対し，リターンキーを押しただけで，BASICプログラムは全く入力されていません．このとき，EB16H番地は18，EB17H番地は86となっています．ここで注意しなければならないのは，ラベル・テーブルのスタート・アドレスは1886H番地ではなく8618H番地となることです．これは，CPUにμPD780を使用しているためで(Z80，8080Aでも当てはまる)，２バイトのデータをメモリ上に置く場合，必ず上位バイトと下位バイトが入れ換わります．それで，CLEARの第２パラメータの下限は，8618H＋

533$_{(10)}$＝882DHとなります．このときのマシン語領域は，E5FFH－882DH＝5DD2Hつまり約24Kバイトということになります．



```
CLEAR ,&H882D
Ok
CLEAR ,&H882C
?OM Error
Ok
```

第2パラメータの下限値は882DHとなる

第5章　マシン語プログラミング

さて，本章ではいよいよマシン語の解説に入ります．前章までの段階で，マシン語の学習に必要な一通りの知識は備わっていますので，本章の内容についても無理なく理解できるでしょう．

本章の学習内容は，マシン語プログラミングの上で，頻繁に使用される命令が実際にどのように使われているかを理解すると共に，基本的なプログラム・テクニックを学ぶことです．ここで登場する命令は，ニーモニックの表現形式によって分類してありますので，体系的に命令を覚えることができます．

なお，本章に出てくるプログラムはすべて入力して実行してください．実行することにより，一層の理解が深まることでしょう．コンピュータ界の名言(迷言?)には，次のようなものがありますので………

「論よりLIST」
「論よりRUN」

5.1　ニーモニックの表現方式

この節では，ニーモニックの表現形式について説明しておきます．表現形式などというと何やら堅苦しい感じがしますが，BASICリファレンス・マニュアルの書式のような一般的な命令の書き方のことをいいます．

ニーモニックの一般的な表現形式は次のようなものです．

［ラベル］：〈オペコード〉［〈第1オペランド〉［，〈第2オペランド〉］］

ニーモニックの表現では，アルファベットはすべて大文字を使用します．また，命令によっては，オペ・コードだけのもの，オペランドを1個もつもの，オペランドを2個もつものの3種類があります．

オペコードのみ	オペランド1個	オペランド2個

LDIR
EXX
CPI
	PUSH HL
	PUSH IX
	POP IY
	OR B
		LD A, B
		LD B, H
		LD C, 40H
		ADD A, L
		BIT 4, A

表5.1　いろいろな命令の例

表5.1におけるLD A,BとLD B,Hは全く同種の命令です．この命令は，オペランドに入るレジスタが変わるだけで49個ありますが，個々の例を挙げて説明することは非効率的なので，この命令のオペランド部分は次のようにシンボルで表現します．

LD r,r′

ここで，rとr′にあてはまるものは次の7種類で，それぞれ8ビットのレジスタ名を表しています．

A,B,C,D,E,H,L

シンボルは，この章およびAPPENDIXのインストラクション表において，いくつかの命令をまとめて紹介するために使われるものですから，まずここで明確に定義しておきましょう．

なお，以後の説明で〝CPU〟と表記されているのは，μPD780を表すことにします．

5.1.1　シンボルのはなし

シンボルは，オペランドの記述を代表する記号です．先に出てきたr,r′を含めて17種類ありますが，プログラム中の命令と併せて考えていけば自然と覚えられるものです．ただ，シンボルの意味を忘れてしまったときは，必ずここに戻って確認するようにしてください．

▶rおよびr′

rおよびr′は，CPUの8ビットレジスタのうち次の7個を表します．

A,B,C,D,E,H,L　各レジスタ

▶dd,ss,qq,pp,およびrr

これらは，16ビットで扱うことができるCPUのレジスタを表すものです．8ビットのレジスタをペアにして使うものと，16ビット単位でしか扱えないものがあります．

●dd および ss

BC,DE,HL 各レジスタ・ペア

SP

●qq

BC, DE, HL, AF 各レジスタ・ペア

●pp

BC, DE 各レジスタ・ペア
IX
SP

●rr

BC, DE 各レジスタ・ペア
IY
SP

▶b

bは，レジスタではなく，b番目のビットを表す数です．また1バイト内のビットを扱いますから，0～7の値でなければなりません．

▶cc

条件分岐命令においての条件を表します．これには次のものがあり，これらの条件が成立するとき指定されたアドレスへ分岐します．また，HフラグおよびNフラグを条件とする分岐命令はありません．

NZ	Zフラグが	0
Z	Zフラグが	1
NC	CYフラグが	0
C	CYフラグが	1
PO	P／Vフラグが	0
PE	P／Vフラグが	1
P	Sフラグが	0
M	Sフラグが	1

▶p

RST命令においての分岐先を表すもので，次の8種類があります．

00H, 08H, 10H, 18H, 20H, 28H, 30H, 38H

▶n

8ビットのデータを表します．

▶nn

16ビットのデータを表します.

▶e

相対分岐命令における8ビットのディスプレイスメントを表し, eの表示は2の補数を使用します.

▶d

8ビットのデータを表します. nと違うのは, この値がインデックス・レジスタのディスプレイスメントとなることです. dの表示は2の補数を使用します.

▶(x)

( ) で囲まれたオペランドは, xのもつアドレスの内容を表します.

(HL)	HLレジスタの示すメモリの内容
(IX+d)	IXレジスタにディスプレイスメントdを加えたメモリの内容
(nn)	nnで示されるメモリの内容
(C)	Cレジスタが指し示すI/Oポートのアドレス
(n)	nで示されるI/Oポートのアドレス

▶s

r , n , (HL) , (IX+d) , (IY+d) を表します.

▶m

r , (HL) , (IX+d) , (IY+d) を表します.

5.1.2　μPD 780にないニーモニック

本書に掲載しているマシン語プログラムは, すべてDUAD-88Dというアセンブラによって作成しています. 実はアセンブラを使用してプログラムを作る場合には, CPUのニーモニックにない命令が出てくるのです. これを擬似命令といいます.

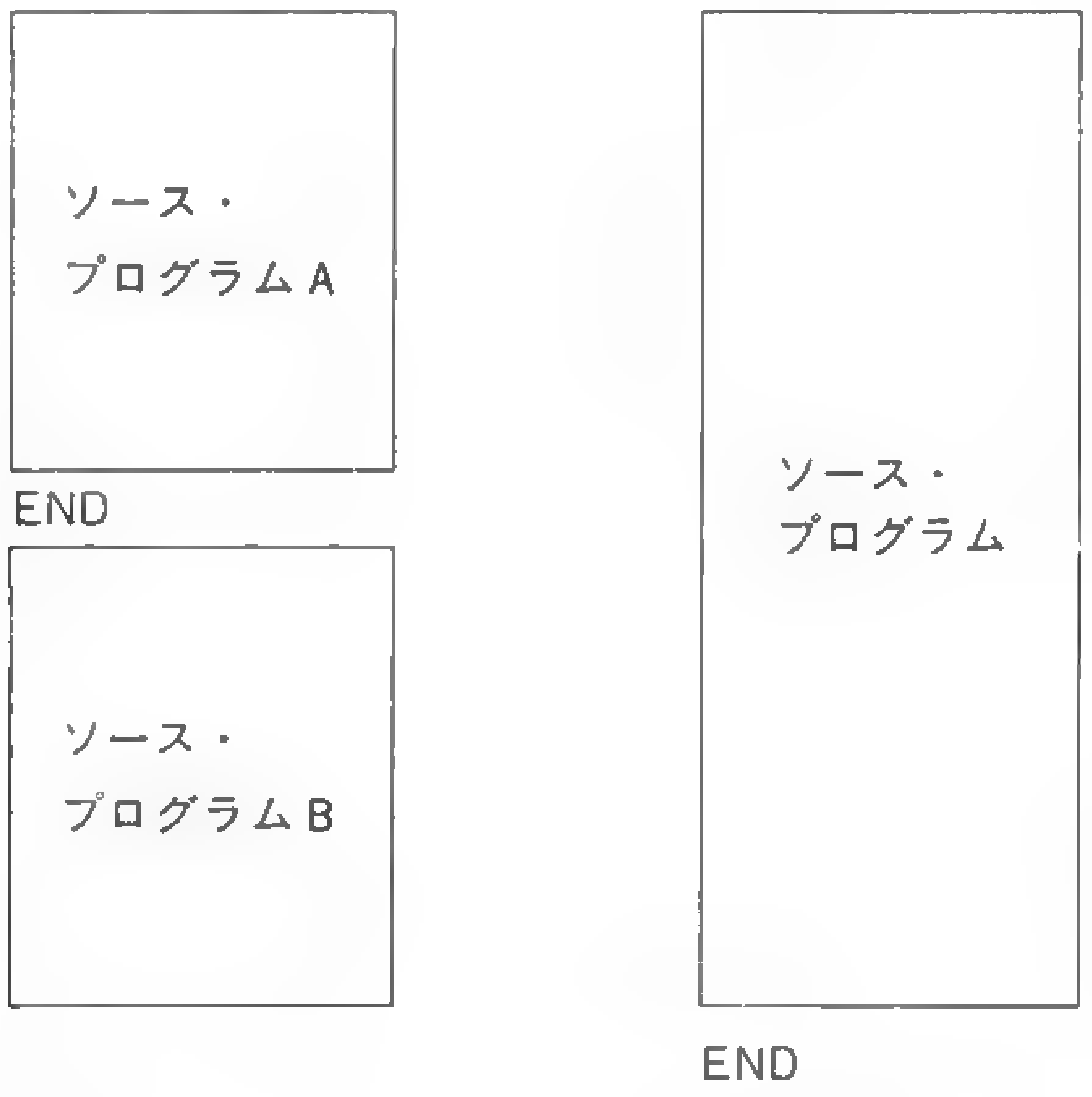
擬似命令は, ハンドアセンブルでプログラミングしている人にとっては無用のものです. しかし, 次節からの命令の説明に出てくるリストは, すべてMF-ASMを使用することを前提としていますので, ここで説明することにします.

擬似命令とは，CPUに対して指示を行う命令ではなく，アセンブラに対して指示を行う命令です．ですから使用するアセンブラの仕様(文法)によって，型式が異なることがあります．

DUAD-88DのアセンブラとMF-ASMには，以下の6種類の擬似命令があります．

EQU　　DB（またはDEFB）
ORG　　DW（またはDEFW）
END　　DS（またはDEFS）

ENDは，BASICにもありましたからだいたい見当がつくと思いますが，ソース・プログラムの終わりを表します．しかし，ここで注意しなければならないのは，BASICではこれがプログラムの実行を終了する命令であったのに対し，ここではアセンブラに対して，ソース・プログラムのいちばん最後を宣言する命令であるということです．図5．1をみてください．



(a) 誤った例　　(b) 正しい例

図5.1　ENDの使用場所

(a)の例をアセンブラに通すと，Aの部分はアセンブルされてマシン語に変換されますが，途中のENDで，アセンブラは，ソース・プログラムの終わりであると判断して処理をやめますから，Bの部分は無視されてしまいます．

EQU(Equate)は，定数にラベルで意味付けするものです．図5．2をみてください．

```
ABC：EQU  100
DEF：EQU  10H
GHI：EQU  ABC+DEF
```

ラベル

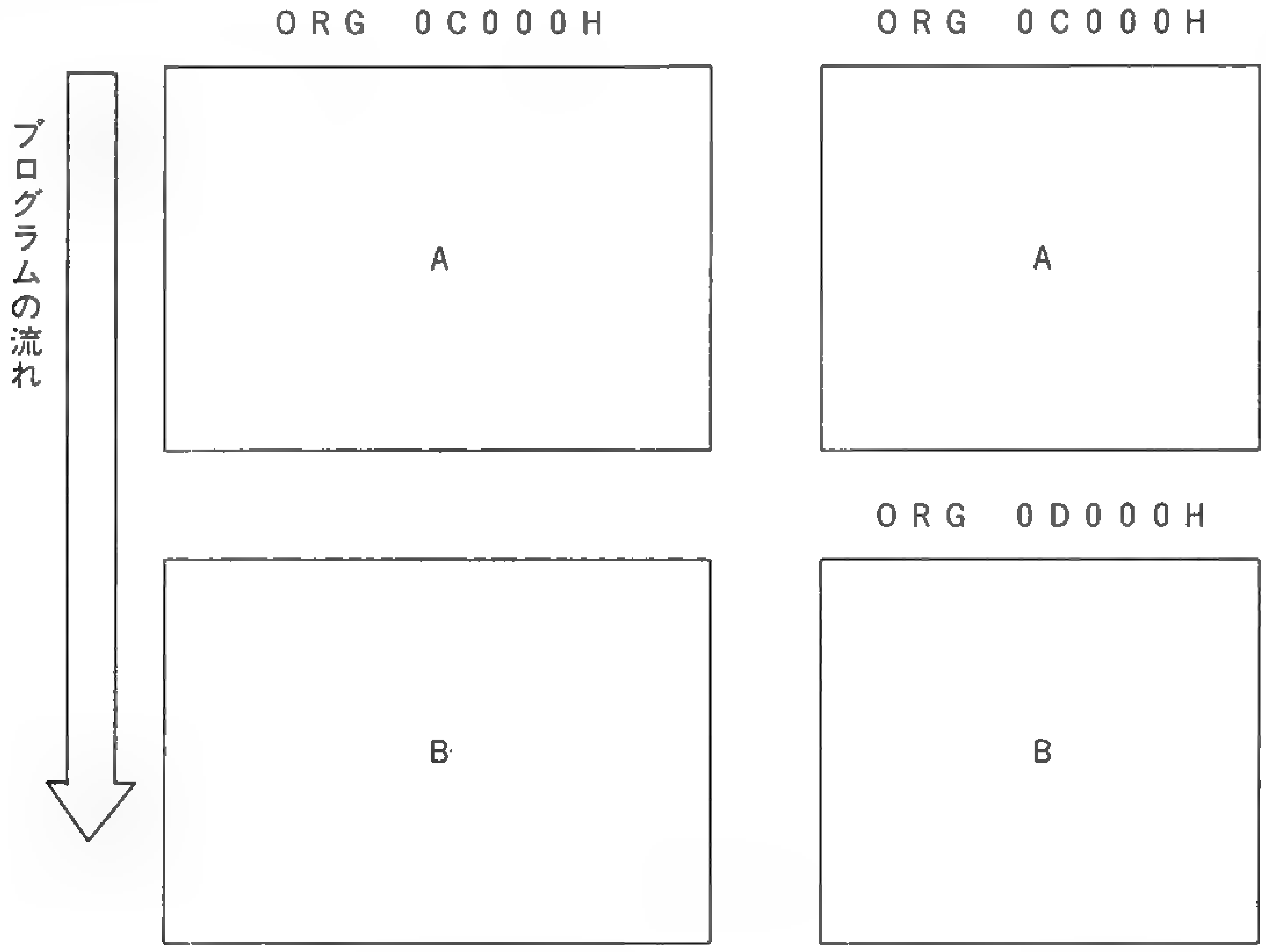
図5.2　EQUの例

1行目と2行目は，それぞれABCに100，DEFに10Hつまり16を定義しています．3行目は，GHIにABC＋DEFの値116を定義したことになります．

このように，数値データなどをEQUでラベルで扱うようにしておくと，プログラム自体が見やすくなりますし，アセンブラを使用しているときにデータの変更があった場合には，EQUの行のみの変更で済みます．

なお，ラベルのABC，DEF，GHIにそれぞれコロン（：）がついているのは，アセンブラの文法上の約束で，ラベルを定義する場合には，必ずコロンをつけることになっています．

ORG（Origin）は，次の行から始まるマシン語プログラムのスタート・アドレスを定義するものです．図5．3をみてください．



（a）ORGが1つの場合　　（b）ORGが2つの場合

図5.3　ORGの例

(a)は，1つのソース・プログラム内に，ORGが1か所しかない例で，この場合Aの終了アドレスの次がBのスタート・アドレスになります．よってAとBは1つのつながったものになります．

これに対して(b)では，Aの終了アドレスがC001H番地であったとしても，Bは，D000H番地から始まります．

DB（Define Byte）は，バイト単位のデータをメモリ上に設定します．文字列の場合にもこの命令を使います．

DW（Define Word）は，ワード単位つまり2バイトのデータをメモリ上に設定します．この場合，上位バイトのデータがアドレスの大きいメモリに，下位バイトがアドレスの小さいメモリに設定

されることに注意してください．

DS(Define Storage)は，必要なバイト数のメモリ領域を確保します．

以上のDB, DW, DSの使用例をリスト１に示します．

リスト1

```
                     ORG  0C000H
              ;
C000 0A              DB   10
C001 10              DB   10H
C002 00010203        DB   0,1,2,3,4,5,6,7,8,9
C006 04050607
C00A 0809
C00C 41614262        DB   'A','a','B','b','C'
C010 43
C011 41424344        DB   'ABCDEFGHIJKL'
C015 45464748
C019 494A4B4C
              ;
C01D 0100            DW   1
C01F FF00            DW   0FFH
C021 D711            DW   4567
C023 6745            DW   4567H
C025 34127856        DW   1234H,5678H,9ABCH,0DEFH
C029 BC9AEF0D
C02D 01000200        DW   1,2,3,4,5,6,7,8,9
C031 03000400
C035 05000600
C039 07000800
C03D 0900
C03F 42414443        DW   'AB','CD','EF','GH'
C043 46454847
              ;
C047                 DS   10
C051                 DS   10H
C061                 DS   15
              ;
C070                 END
```

実際にプログラムの中でどのように使われるのかは，後出するリストの中でみていただきたいと思います．

5.2　マシン語ツール

単純な四則演算の計算処理を行う場合でも，紙と鉛筆で計算するのと電卓で行うのではその処理速度には大きな違いがあります．つまり，どのようなツールを利用するかの違いは，仕事の効率を大きく左右するのです．

マシン語のプログラム開発について考えれば，ハンドアセンブルでの開発とアセンブラを利用して行う場合では，効率的には雲泥の差が生じることは明らかだと思います．

例えば，本書に掲載されているプログラムを実行するに当たっては，そのプログラムを入力する場合にモニタ・コマンドを用いることができます．しかし16進コードを意味もなく入力することに果して意義があるのでしょうか．ゲーム・プログラムの入力であれば，そのプログラムの実行結果を得るという目的ですから，その作業が多少めんどうな作業であっても仕方はありません．しかし学習を目的とする場合には，BASICのコマンドを覚えるにもプログラムを入力して覚えることが一番効果的であるのと同様に，ソース・プログラムを入力していった方が命令を覚えるためには効果的であることは明らかだと思います．またアセンブラの使用環境に慣れておくことは，今後のプログラム開発に向けて有益なことだと思われます．

本章からのマシン語の学習では，読者の皆さんがMF-ASMを利用されることを前提としています．ですから本章を読まれる前に，APPENDIXのMF-ASMの項を参照し，その使用法を理解されておくことが望ましいでしょう．

5.3　メモリの内容交換(1)

いよいよ命令の説明に入ります．この節では3つのマシン語プログラムのリストが登場していますが，どれもテキストVRAM上のキャラクタ座標（0，0）と（1，0）にあたるメモリ・アドレスの内容を交換するものです．しかし，結果は同じでも，求める手法は異なっていますので，その違いを理解して，どのリストが最適かを考えてください．

リスト2

```
                   ;
                   ;     SAMPLE  LD  1
                   ;
F3C8               VRAM:  EQU  0F3C8H      テキストVRAMの先頭アドレス
F3C9               VRAMP: EQU  VRAM+1      テキストVRAMの先頭アドレスの次のアドレス
                   ;
                          ORG  0E000H
                   ;
E000 3AC8F3               LD   A,(VRAM)    Aにキャラクタ座標（0，0）の内容をロードする
E003 47                   LD   B,A         BにAをロードする
E004 3AC9F3               LD   A,(VRAMP)   Aにキャラクタ座標（1，0）の内容をロードする
E007 32C8F3               LD   (VRAM),A    キャラクタ座標（0，0）に（1，0）の内容を出力する
E00A 78                   LD   A,B
E00B 32C9F3               LD   (VRAMP),A   キャラクタ座標（1，0）に（0，0）の内容を出力する
                   ;
E00E FF                   RST  38H         モニタへ戻る
                   ;
E00F                      END
```

リスト3

```
                ;
                ;    SAMPLE   LD   2
                ;
F3C8            VRAM:   EQU   0F3C8H     テキストVRAMの先頭アドレス
F3C9            VRAMP:  EQU   VRAM+1     テキストVRAMの先頭アドレスの次のアドレス
                ;
                        ORG   0E000H
                ;
E000 21C8F3             LD    HL,VRAM    HLにテキストVRAMの先頭アドレスをセットする
E003 11C9F3             LD    DE,VRAMP   DEに次のアドレスをセットする
E006 46                 LD    B,(HL)     Bにキャラクタ座標（0，0）の内容をロードする
E007 1A                 LD    A,(DE)     Aにキャラクタ座標（1，0）の内容をロードする
E008 77                 LD    (HL),A     キャラクタ座標（0，0）に（1，0）の内容を出力する
E009 78                 LD    A,B
E00A 12                 LD    (DE),A     キャラクタ座標（1，0）に（0，0）の内容を出力する
                ;
E00B FF                 RST   38H        モニタへ戻る
                ;
E00C                    END
```

リスト4

```
                ;
                ;    SAMPLE   LD   3
                ;
F3C8            VRAM:   EQU   0F3C8H     テキストVRAMの先頭アドレス
                ;
                        ORG   0E000H
                ;
E000 DD21C8F3           LD    IX,VRAM    IXにテキストVRAMの先頭アドレスをセットする
E004 DD4600             LD    B,(IX)     Bにキャラクタ座標（0，0）の内容をロードする
E007 DD4E01             LD    C,(IX+1)   Cにキャラクタ座標（1，0）の内容をロードする
E00A DD7100             LD    (IX),C     キャラクタ座標（0，0）に（1，0）の内容を出力する
E00D DD7001             LD    (IX+1),B   キャラクタ座標（1，0）に（0，0）の内容を出力する
                ;
E010 FF                 RST   38H        モニタへ戻る
                ;
E011                    END
```

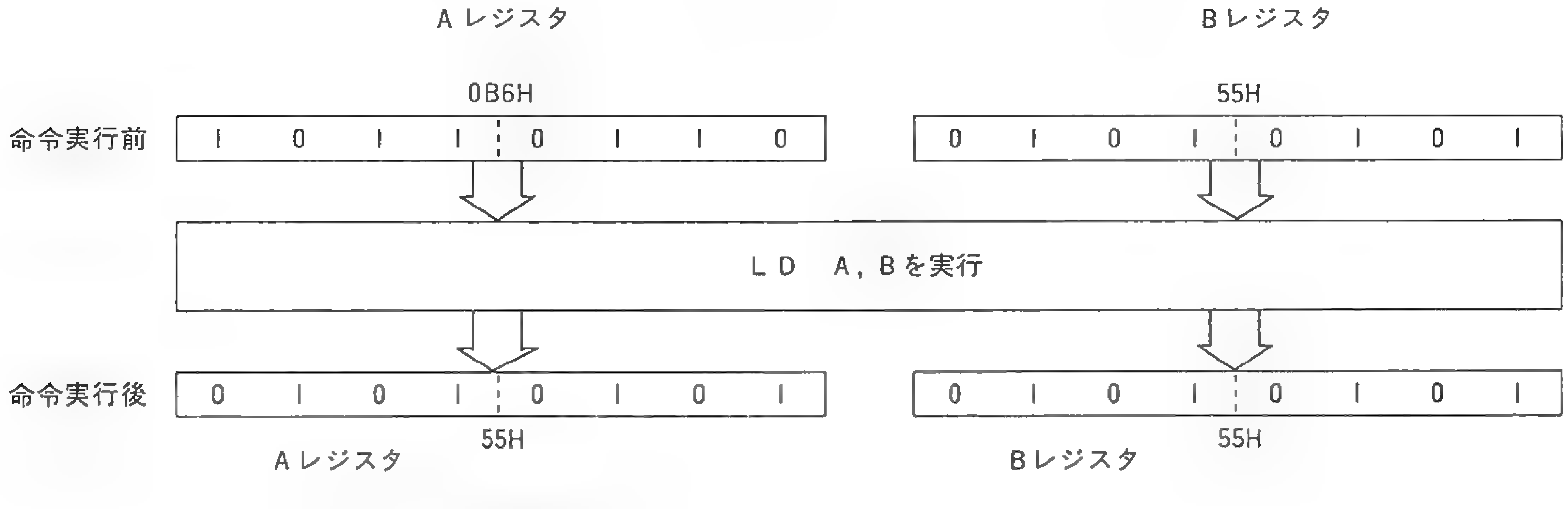
5.3.1　LD命令

リスト２～４は，基本的にはLD命令のみで構成されています．LD命令のLDは，LOADを略したものですが，BASICのLOADとは意味が異なります．BASICではカセットテープやフロッピー・ディスクにセーブされたプログラムを本体に取り込むためのコマンドですが，マシン語でのLDは，LOADよりはむしろLETに近いものといえます．

次の命令を例として考えてみましょう．

```
LD      A,B
```

この命令は，Bレジスタの内容をAレジスタに移すものです（データを移すことをロードという）．図5．4をみてください．



Aレジスタには Bレジスタの内容がロードされるが
Bレジスタの内容は変わらない

図5.4　ロード命令の働き

ニーモニック	命令の動作	命令のサンプル	
		ニーモニック	マシン語
LD A,(nn)	Aレジスタにnnで示されるメモリの内容をロードする	LD A,(1234H)	3A 3412
LD r,r'	r'レジスタの内容をrレジスタにロードする	LD A,B	78
LD (nn),A	LD A,(nn)の逆の命令．つまりnnで示されるメモリにAレジスタの内容をロードする	LD (1234H),A	32 3412
LD dd,nn	ddで示されるレジスタ・ペアまたは SP（スタック・ポインタ）に，16ビットデータnnをロードする	LD BC,5732H	01 3257
LD r,(HL)	rレジスタに，HLレジスタ・ペアで示されるメモリの内容をロードする	LD B,(HL)	46
LD (HL),r	LD r,(HL)の逆の命令.HLレジスタ・ペアで示されるメモリにrレジスタの内容をロードする	LD (HL),C	71
LD A,(DE)	AレジスタにDEレジスタ・ペアで示されるメモリの内容をロードする．同種の命令にLD A,(BC)がある	LD A,(DE) LD A,(BC)	1A 0A
LD (DE),A	DEレジスタ・ペアで示されるメモリにAレジスタの内容をロードする．同種の命令にLD (BC),Aがある	LD (DE),A LD (BC),A	12 02
LD IX,nn	IXレジスタに16ビットの値nnをロードする．同種の命令にLD IY,nnがある	LD IX,1234H	DD 21 3412
LD r,(IX+d)	rレジスタにIXレジスタの示す値にディスプレイスメントdを足した値をアドレスとするメモリの内容をロードする．なお，ディスプレイスメントdが0の場合は+dを省略して(IX)と表記してもよいことになっている.同種の命令に，LD r，(IY+d)がある	LD B,(IX+1) LD A,(IX)	DD 46 01 DD 7E 00

LD (IX+d),r	LD r,(IX+d)の逆の命令で，IXレジスタにディスプレイスメントdを加えた値をアドレスとするメモリにレジスタrの内容をロードする．同種の命令にLD (IY+d),rがある	LD (IX−128),A
		LD (IX+127),C
		LD (IX+10H),H
			DD 77 80
			DD 79 7F
			DD 74 10

RST 38H	モニタのコマンド・レベルに戻る		

命令のまとめ1

これはLD命令の働きを図に示したものです．命令の実行前はAレジスタとBレジスタの内容は異なっていますが，実行後はAレジスタの内容とBレジスタの内容は等しくなっていますから，これはBASICの次の命令とよく似ています．

LET A＝B

このように，LD命令は代入文としての働きをもっており，以下のようなバリエーションがあります．

命令の動作	ニーモニック
(1)　レジスタの内容を他のレジスタへ移す	LD r, r′
(2)　レジスタに任意の値を直接代入する	LD r, n;LD dd, nn
(3)　任意のメモリ内容をレジスタに移す	LD r, (HL);LD dd, (nn)
(4)　レジスタの内容を指定されたメモリに移す	LD (HL), r;LD (nn), dd

LD命令は，マシン語のプログラムの中では最もポピュラーな命令であり，BASICでいわれる基本ステートメントにあたるものですから，どのようなものがあるかよく理解しておいてください．

5.3.2　メモリの入れ換え

LD命令の意味が理解できたと思いますので，リストの説明に入ります．まずリスト2をみてください．この場合VRAMというラベルにはF3C8Hという値を定めていますが，この値はキャラクタ座標（0，0）にあたるメモリ・アドレスです．VRAMPはVRAM＋1の値ですから，F3C9Hになります．またORG 0E000Hとありますから，このプログラムのスタート・アドレスはE000H番地となります．実際のマシン語プログラムは，次のLD A,(VRAM)から始まり，これはAレジスタにメモリ・アドレスVRAMの内容を移すもので，F3C8H番地のメモリの内容がAレジスタに入ります．次に，Aレジスタの内容をBレジスタに移し，VRAMPの内容をAレジスタに移して，これをVRAMに移します．この時点で，キャラクタ座標（1，0）の内容が（0，0）に移ったことになります．そして，先程Bレジスタに保存しておいたキャラクタ座標（0，0）の内容をAレジスタに移した後，VRAMP，つまりキャラクタ座標（1，0）に移して交換が終了します．最後のRST 38Hは，実行終了後にモニタに戻る命令です．

いかがですか，短いサンプルですからリストだけでも充分理解できると思いますが，やはり「百聞は一見に如かず」です．実行してみましょう．

5.3.3　チャレンジ

すでにMF-ASMについてはAPPENDIXの部分で理解されていると思いますが，初めてのチャレンジですので一通りの手続きを追っていきます．

■ プログラムの入力

最初にMF-ASMの領域を確保しますので，BASICモードで次の命令を実行してください．

```
CLEAR ,&HB8FF↵
```

次にMF-ASMをロードします．テープとディスクの場合では方法が異なりますので注意してください．

テープの場合，モニタのRコマンドを使用します．

```
h]Rファイル・ネーム↵
```

ディスクの場合はBASICのBLOADを使用します．

```
BLOAD "ファイル・ネーム"↵
```

MF-ASMのロードが終了したら，モニタ・モードのGコマンドでMF-ASMのイニシャライズを行い，いったんBASICモードに戻ります．イニシャライズ・プログラムはB900H番地から置かれています．

```
h]GB900↵
h]∧B
```

ソース・プログラムを入力し，CMDをダイレクトステートメントで実行してください．これでアセンブル作業は終了します．

■ プログラムの実行

アセンブル作業が終了した時点で，画面上ではオプション・コマンドの入力待ちになっていると思います．

```
OPTION is "l p s o e"
Option  ?  o↵
```

この時点でoコマンドを入力すると次のように表示されますが，ここでは単にリターンキーを押してください．

```
LOAD OFFSET  ?↵
```

次にBASICとモニタのモードの選択を聞いてきますので，Bを入力してBASICモードに戻ります．

```
RETURN TO BASIC OR MONITOR(B/M)  ?B↵
```

リスト2～4は，テキストVRAMのメモリ交換を行っていますので，画面に何も表示されていなければ実際に走らせてみても何をやっているのかわかりません．そこで画面の設定を行います．リスト5をみてください．

リスト5

```
100 '
110 '   set screen
120 '
130 CLEAR ,&HDFFF
140 KEY 1,"ge000"+CHR$(13)
150 WIDTH 80,25 : CONSOLE 1,23
160 LOCATE 0,0
170 PRINT "@ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklm";
180 PRINT "nopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ@"
190 LOCATE 20,24
200 PRINT "0123456789012345678901234567890123456789"
210 MON
```

CRTの最上段と最下段に英数字を表示し，f・1 に“ge000”を定義するプログラムです．このプログラムは，後出するサンプル・プログラムでも利用しますので，入力したらセーブしておいてください．

リスト5が入力できたら実行してください．すでに，プログラム中でモニタ・モードに入っているので，f・1 を押します．

いかがですか，テキストVRAMのメモリの内容が入れ換わっていることが確認できたと思います．リスト3，4についての説明は省略しますが，リストのコメントと命令の動作を参考にしていただければ理解できると思います．

ここで注意しなければならないことを挙げましょう．

```
LD      A, (nn)
```

この命令のマシン語はオペランドとして16ビットのアドレスを持ち，図5.5の例のように，マシン語のオペランドは前にも述べた通りアドレスの下位8ビットが先に，上位8ビットが後になります．

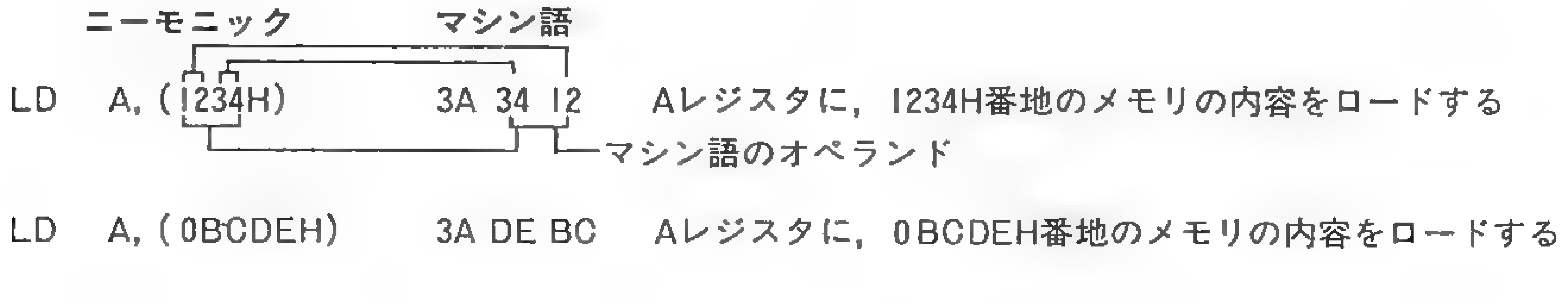

ニーモニック
マシン語
LD　A, (1234H)
3A 34 12
Aレジスタに，1234H番地のメモリの内容をロードする
マシン語のオペランド
LD　A, (0BCDEH)
3A DE BC
Aレジスタに，0BCDEH番地のメモリの内容をロードする

図5.5　上位と下位の入れ換え

```
RST     38H
```

この命令については，5.7.3のCALLの項で説明しますのでそちらを参照してください．ここでは，単純にモニタのコマンド・レベルに戻る命令だと思っていただいてかまいません．

マシン語プログラムの場合，どのようなプログラムでも実行終了後になんらかの処理が必要です．このリスト2の例だと，RST 38Hがなければ，CPUはE00EH番地にあるメモリの内容を勝手に命令として取り込み，実行しようとします．これはCPUの暴走につながりますから，マシン語プログラムの場合，必ず最後にモニタへ戻るかBASICへ戻る処理が必要です．

5.4 メモリの内容交換(2)

LD命令では多くのバリエーションが登場しましたが，皆さん覚えられたでしょうか？最初からがむしゃらに16進コードを覚える必要はありません，16進コードへの変換はアセンブラにまかせて，皆さんは命令の意味とプログラムの流れを把握するようにしてください．

さて，本節のサンプル・プログラムは前節のものを少々アレンジしてみました．リスト6の実行結果は，5.3で説明したリスト2～4となんら変わりありません．ただ，次のニーモニックが新しく加わっています．

```
EX      DE,HL
```

5.4.1 EX命令

EX命令は，第1オペランドと第2オペランドで指定されたレジスタまたはメモリの内容を交換する交換命令です．また，この命令はオペランドが非常に限定されており，全部で6種類しかありません．BASICの命令ではEX命令と似たものがあるでしょうか．皆さんはもう気が付いているでしょう．そうSWAP！

ニーモニック	命令の動作	命令のサンプル	
		ニーモニック	マシン語
EX DE,HL	DEレジスタ・ペアとHLレジスタ・ペアの内容，つまりHレジスタとDレジスタ，EレジスタとLレジスタの内容を交換する	EX DE,HL	EB
EXX	汎用レジスタの主レジスタと副レジスタをまとめて交換する	EXX	D9
EX (SP),HL	SPが指し示すメモリの内容とLレジスタの内容，SP+1のメモリの内容とHレジスタのメモリの内容を交換する	EX (SP),HL	E3
EX AF,AF'	AFレジスタ・ペアとAF'レジスタ・ペアの内容を交換する	EX AF,AF'	08
EX (SP),IX	SPが指し示すメモリの内容とIXレジスタの下位8ビット，SP+1が指し示すメモリの内容とIXレジスタの上位8ビットの内容を交換する．同種の命令にEX (SP),IYがある	EX (SP),IX	DD E3

命令のまとめ2

5.4.2　リストの実行にあたって

ここではリストの説明は前節とほとんど同じですので省略させていただきます．ただ，プログラムの実行に際して注意を1つ．画面設定のプログラム（リスト5）が必要です．

リスト6

```
                 ;
                 ;    SAMPLE    EX
                 ;
F3C8             VRAM:  EQU   0F3C8H    テキストVRAMの先頭アドレス
F3C9             VRAMP: EQU   VRAM+1    テキストVRAMの先頭アドレスの次のアドレス
                 ;
                        ORG   0E000H
                 ;
E000 21C8F3             LD    HL,VRAM   HLにテキストVRAMの先頭アドレスをセットする
E003 11C9F3             LD    DE,VRAMP  DE=HL+1(テキストVRAMの先頭アドレスの次のアドレス)
E006 1A                 LD    A,(DE)    AにDEで示されたメモリの内容をロードする
E007 4E                 LD    C,(HL)    CにHLで示されたメモリの内容をロードする
E008 EB                 EX    DE,HL     DEとHLを交換する        }
E009 71                 LD    (HL),C    HLで示すメモリへCを出力する } メモリの内容を入れ換える
E00A 12                 LD    (DE),A    DEで示すメモリへAを出力する }
                 ;
E00B FF                 RST   38H       モニタへ戻る
                 ;
E00C                    END
```

5.5　メモリの内容交換(3)

メモリの内容を入れ換えるのにも様々な方法があり，この節ではいよいよ第3弾をお送りします．

さて，前節までは単に2つのメモリ内容を入れ換えるだけでしたが，ここでは少し規模を大きくして，1行そっくり入れ換えてみようと思います．リスト7，8を見較べると異なった命令を使っているのがわかりますが，どちらも同じ結果が得られるもので，キャラクタ座標の(n，0)と(n，24)のメモリ内容を交換するものです．つまり画面の最上段の行と最下段の行を入れ換えているのです．

リスト7

```
                 ;
                 ;    SAMPLE    DEC   JR
                 ;
F3C8             VRAM:  EQU   0F3C8H    テキストVRAMの先頭アドレス
FF08             VBOTOM:EQU   0FF08H    キャラクタ座標（0，24）のテキストVRAMのアドレス
                 ;
                        ORG   0E000H
                 ;
E000 21C8F3             LD    HL,VRAM   HLにテキストVRAMの先頭アドレスをセットする
E003 1108FF             LD    DE,VBOTOM DEにキャラクタ座標（0，24）のアドレスをセットする
E006 0650               LD    B,80      Bに繰り返し回数をセットする
```

```
                    ;
E008 4E             LOOP:  LD    C,(HL)       HLで示されるメモリ(キャラクタ座標(n, 0))の内容をCにロードする
E009 1A                    LD    A,(DE)       DEで示されるメモリ(キャラクタ座標(n, 24))の内容をAにロードする
E00A 77                    LD    (HL),A      )
E00B 79                    LD    A,C         }HLで示されるメモリ内容とDEで示されるメモリ内容を交換する
E00C 12                    LD    (DE),A      )
E00D 23                    INC   HL
E00E 13                    INC   DE
E00F 05                    DEC   B            B=B-1
E010 20F6                  JR    NZ,LOOP      もしB≠0ならLOOPへ飛ぶ
                    ;
E012 FF                    RST   38H          モニタへ戻る
                    ;
E013                       END
```

リスト8

```
                    ;
                    ;    SAMPLE   INC DJNZ
                    ;
F3C8                VRAM:  EQU   0F3C8H       テキストVRAMの先頭アドレス
FF08                VBOTOM:EQU   0FF08H       キャラクタ座標（0，24）のテキストVRAMのアドレス
                    ;
                           ORG   0E000H
                    ;
E000 21C8F3                LD    HL,VRAM      HLにテキストVRAMの先頭アドレスをセットする
E003 1108FF                LD    DE,VBOTOM    DEにキャラクタ座標(0，24)のテキストVRAMのアドレスをセットする
E006 0650                  LD    B,80         Bに繰り返し回数をセットする
                    ;
E008 4E             LOOP:  LD    C,(HL)       CにHLで示されるメモリ(キャラクタ座標(n, 0))の内容をロードする
E009 1A                    LD    A,(DE)       AにDEで示されるメモリ(キャラクタ座標(n, 24))の内容をロードする
E00A 77                    LD    (HL),A      )
E00B 79                    LD    A,C         }HLで示されるメモリの内容とDEで示されるメモリの内容を交
E00C 12                    LD    (DE),A      ) 換する
E00D 23                    INC   HL
E00E 13                    INC   DE
E00F 10F7                  DJNZ  LOOP         もしB≠0ならLOOPへ飛ぶ
                    ;
E011 FF                    RST   38H          モニタへ戻る
                    ;
E012                       END
```

リストをみてもわかる通り，初めて登場する命令がいくつかありますので，最初に説明しておきましょう．

5.5.1　INC命令とDEC命令

リスト7の枠で囲んだ部分を見るとINC HL／INC DE／DEC Bと続けて登場しています．これらは増／減命令と呼ばれ，INCはメモリやレジスタの値に1を足す働きをする命令で，DECはメモリやレジスタの値から1を引く働きをする命令です．INC命令もDEC命令も非常に簡単な命令ですが，どちらもLD命令と並んでマシン語の基本となるものですから決して忘れないでください．なお，INCとDECの命令はレジスタが8ビットの場合と16ビットの場合ではフラグの動作が異なりますので，APPENDIXのインストラクション表でその動きを確認してください．

５.５.２　JR命令

リスト７の枠で囲んだ最後の行をみてください．

```
JR      NZ,LOOP
```

この命令のJR NZの部分は，Jump Relative if Non Zeroの省略されたものです．ジャンプレラティブは相対分岐と訳されますが，何が相対的なのかといえば，分岐先のアドレス指定がプログラム・カウンタの指すアドレスを基準として相対的に決まるということです．相対の反対の意味で絶対という言葉がありますが，E008H番地と分岐先を直接指定した場合は，これを絶対アドレス指定といい，E010H番地から数えて９番地前と指定するのが相対アドレス指定なのです．

NZはコンディション（条件）と呼ばれるもので，シンボルではccで表されます．５.１.１シンボルのはなしのところでも述べましたが，この条件が成立したとき（NZはZフラグ＝０）指定されたアドレス（この場合はLOOPで指定されている）へ分岐します．分岐先を示すオペランドはディスプレイスメントeで表されますが，マシン語においてとる値については図５.６を参照してください．

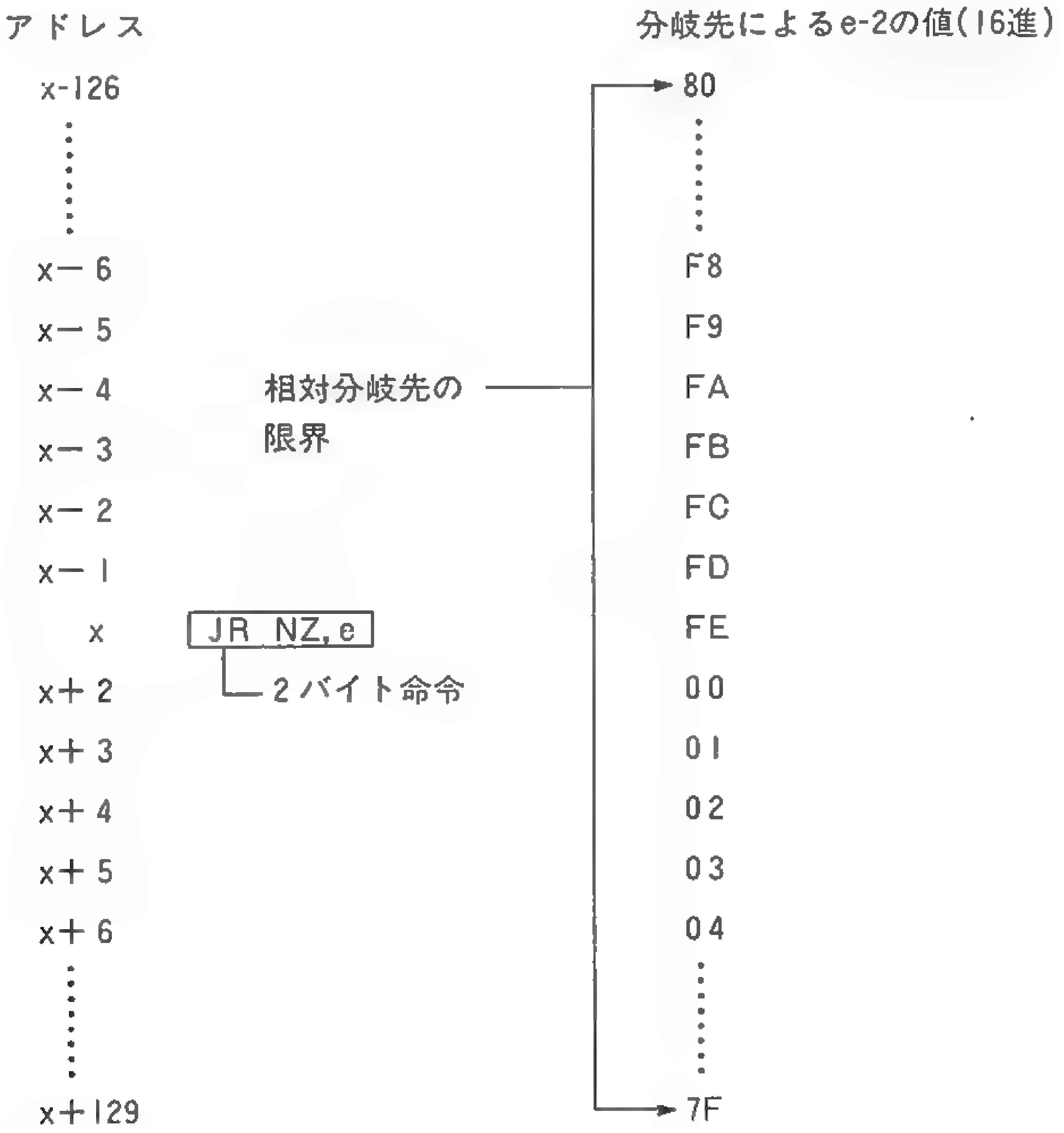

アドレス
x-126
x－6
x－5
x－4
x－3
x－2
x－1
x
x＋2
x＋3
x＋4
x＋5
x＋6
x＋129
相対分岐先の限界
JR NZ,e
2バイト命令
分岐先によるe-2の値(16進)
80
F8
F9
FA
FB
FC
FD
FE
00
01
02
03
04
7F

図5.6　相対アドレスの分岐先

５.５.３　DJNZ命令

リスト７とリスト８を較べると，リスト７でDEC B／JR NZ,LOOPとあった部分が，リスト８ではDJNZ LOOPとなっています．この２つのプログラムからは全く同様の結果が得られるということは，DJNZ LOOPは，２つの命令を複合した命令のように考えられます．実際にDJNZは，

DEC BとJR NZ,LOOPを同時に行う命令と考えても，一向に差し支えありませんが，両者の間には多少の違いがあります．

DEC BとJR NZを組み合わせてプログラムを組んだ場合は，JR NZという命令は，DEC Bによって変化したZフラグの状態を判断して分岐する条件分岐命令であったのに対し，DJNZはBレジスタをデクリメント(DEC)してBレジスタの内容そのものが0になっているかどうかを判断して分岐する特殊分岐命令であるという点です．つまり，リスト7ではフラグがプログラムの実行前と実行後では変化しますが，リスト8では全く変化しません．これについては，モニタのXコマンドを使用してもらえばわかりますが，DJNZ eは，プログラム中でループを作る場合，非常によく使用されるものです．

ニーモニック	命令の動作	命令のサンプル	
		ニーモニック	マシン語
LD r,n	レジスタrに8ビットのデータnをロードする	LD A,56H	3E 56
INC ss	ssで指定されるレジスタ・ペアおよびSPの内容をインクリメント(1だけ増やす)する	INC HL	23
DEC ss	INC ssの逆の命令で，ssで指定されるレジスタ・ペアおよびSPの内容をデクリメント(1だけ減らす)する	DEC HL	2B
INC r	rレジスタの内容に1を足す	INC B	04
DEC r	rレジスタの内容から1を引く	DEC B	05
JR NZ,e	Zフラグによって分岐（ジャンプ）する条件分岐命令．つまりZフラグがリセットされているとき，PCの内容にe−2を足した値で示すアドレスへ分岐する．Zフラグがセットされているときは次の命令に進む	JR NZ, e	20 e-2
DJNZ e	Bレジスタの内容をデクリメントして，Bレジスタの内容が0でなければPCの内容にe-2を足した値で示すアドレスへ分岐する	DJNZ e	10 e-2

命令のまとめ3

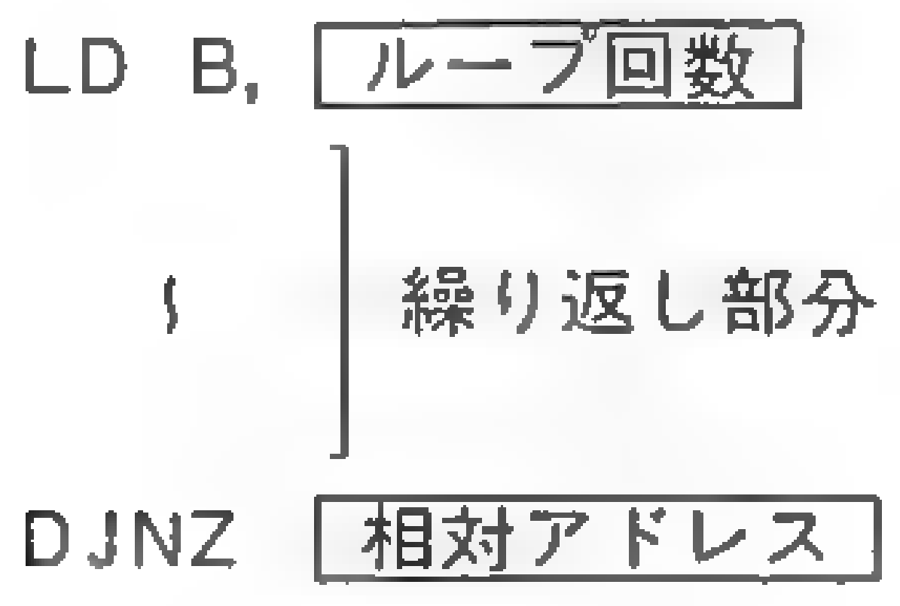


図5.7　DJNZ命令の働き

5.5.4　ラインの入れ換え

新しい命令の説明が終わったところで実際の使い方をみてみましょう．

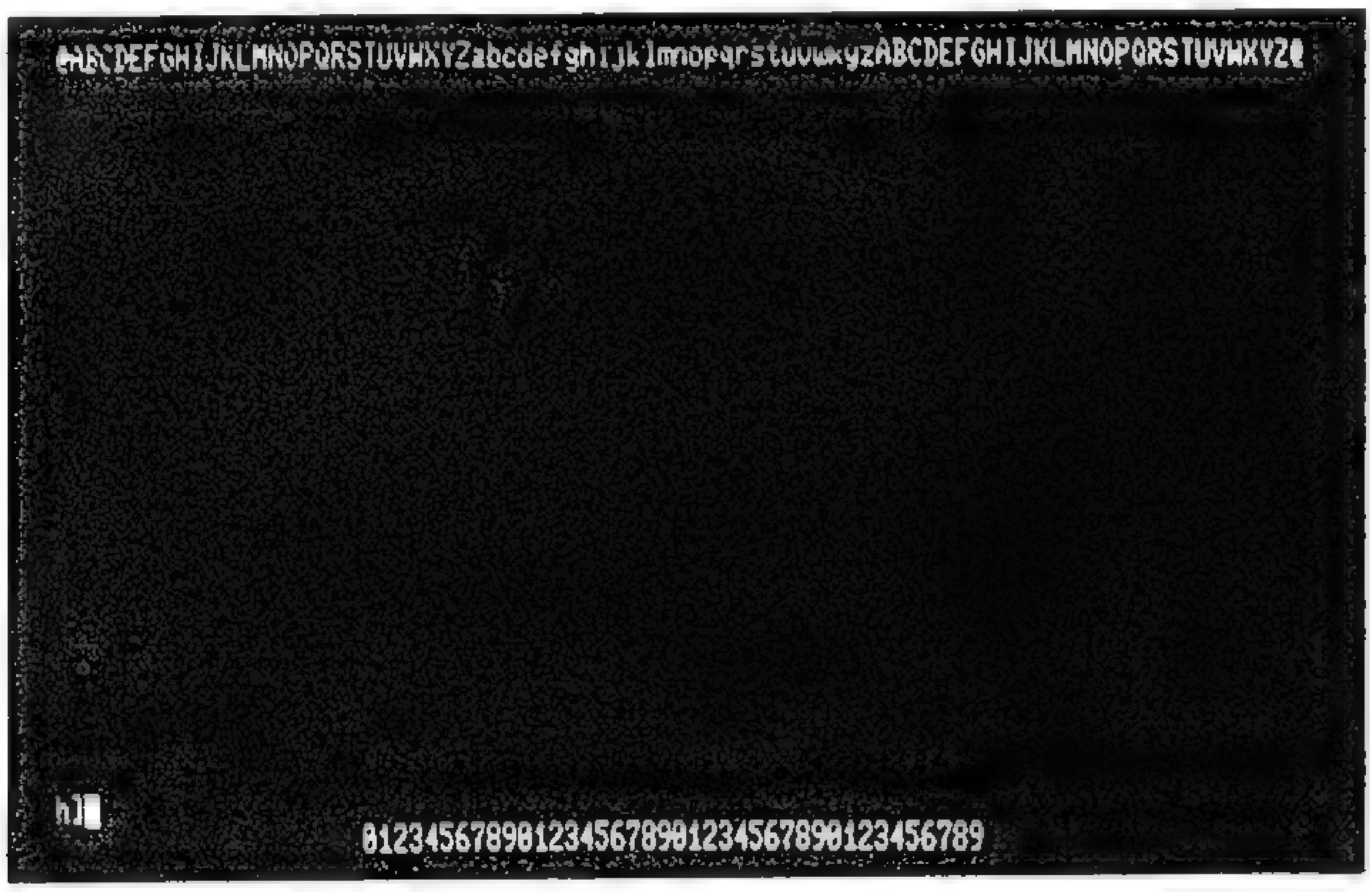
リスト7は，HLレジスタ・ペアにキャラクタ座標（0，0）のメモリ・アドレスを，DEレジスタ・ペアにキャラクタ座標（0,24）のメモリ・アドレスをロードし，リスト2の例と同様にしてメモリの内容を交換します．この後，HL，DE各レジスタ・ペアの値を1ずつ増やし，Bレジスタの値を1減らし，Bレジスタが0でなければLOOPで指定されたアドレスへ分岐します．以上の操作をBレジスタが0になるまで続けます．Bレジスタには，あらかじめ80(WIDTH 80,25の横文字数)

がセットされていましたから，80回つまり1行分の交換を繰り返してプログラムは終了します．

リスト8のプログラムの流れも大筋ではリスト7と同じです．しかしループを形成するのにDJNZを使っているために，DEC Bが省かれています．

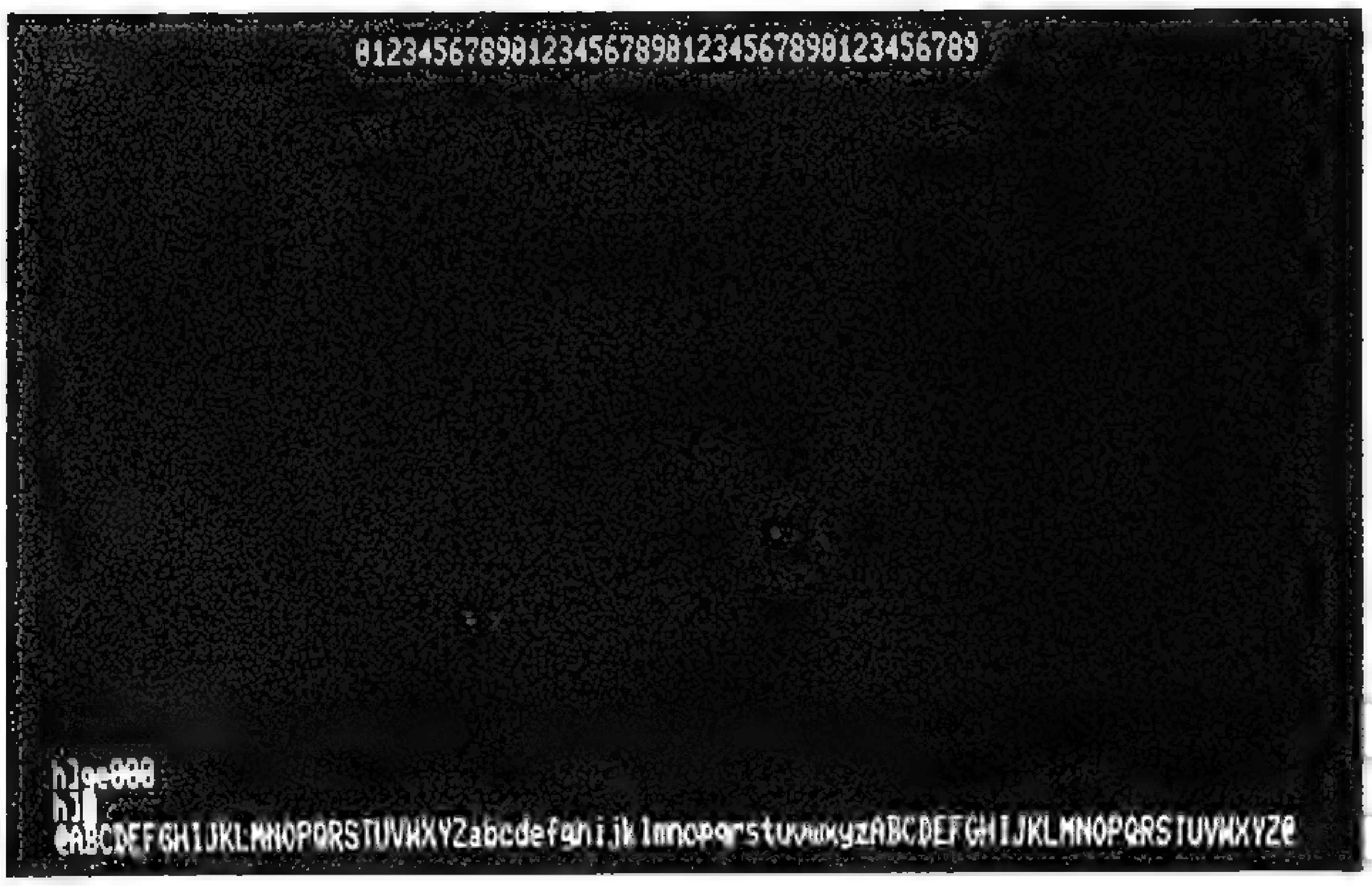
5.5.5　チャレンジ

実際にリスト7を入力して確認しておきましょう．ソース・プログラムを入力してアセンブルを行います．oコマンドでオブジェクト・プログラムをロードしたらBASICモードに戻りリスト5をロードしてください．先程セーブされなかった方は，どうぞもう一度入力してください．準備OKならRUN. f・1 を押せば，実行結果をみることができます．



```
@ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ@









h]█                      0123456789012345678901234567890123456789
```

リスト5の画面設定



```
                         0123456789012345678901234567890123456789








h]ge000
h]█
@ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ@
```

リスト7の実行例．画面設定を変えたり，テキストVRAMの値を変えてみてください

最後のサンプルリストをみてみましょう．

リスト9は，テキストVRAMの最終アドレスをDEレジスタ・ペアに最初にセットして，DEレジスタ・ペアをデクリメントします．この動作により，キャラクタ座標(n,0)と(79−n,24)のメモリ内容が入れ換わります．

リスト9

```
                     ;
                     ;    SAMPLE   INC   DJNZ   DEC
                     ;
F3C8                 VRAM:   EQU    0F3C8H    テキストVRAMの先頭アドレス
FF57                 VEND:   EQU    0FF57H    テキストVRAMの最終アドレス
                     ;
                             ORG    0E000H
                     ;
E000 21C8F3                  LD     HL,VRAM   HLにテキストVRAMの先頭アドレスをセットする
E003 1157FF                  LD     DE,VEND   DEにテキストVRAMの最終アドレスをセットする
E006 0650                    LD     B,80      Bに繰り返し回数をセットする
                     ;
E008 4E             LOOP:    LD     C,(HL)    CにHLで示されるメモリ(キャラクタ座標(n, 0))の内容をロードする
E009 1A                      LD     A,(DE)    AにDEで示されるメモリ(キャラクタ座標(79-n,24))の内容をロードする
E00A 77                      LD     (HL),A   ⎫
E00B 79                      LD     A,C      ⎬HLで示されるメモリの内容とDEで示されるメモリの内容を交換
E00C 12                      LD     (DE),A   ⎭する
E00D 23                      INC    HL        メモリ・アドレスをインクリメントする
E00E 1B                      DEC    DE        メモリ・アドレスをデクリメントする
E00F 10F7                    DJNZ   LOOP      もしB≠0ならばLOOPへ飛ぶ
                     ;
E011 FF                      RST    38H       モニタへ戻る
                     ;
E012                         END
```

リスト9とリスト8の違いは2バイトです．実行結果をみれば明らかですが，2バイト変えるだけでリスト8とこれだけ違うことができるというのは驚きです．まさにマシン語の小回りのよさを物語っています．

なお，チャレンジは皆さんにお任せします．

5.6 キャラクタで線を引く

リスト10は，キャラクタ座標の中央(40, 0)～(40,24)に〝*〟で縦線を引くものです．

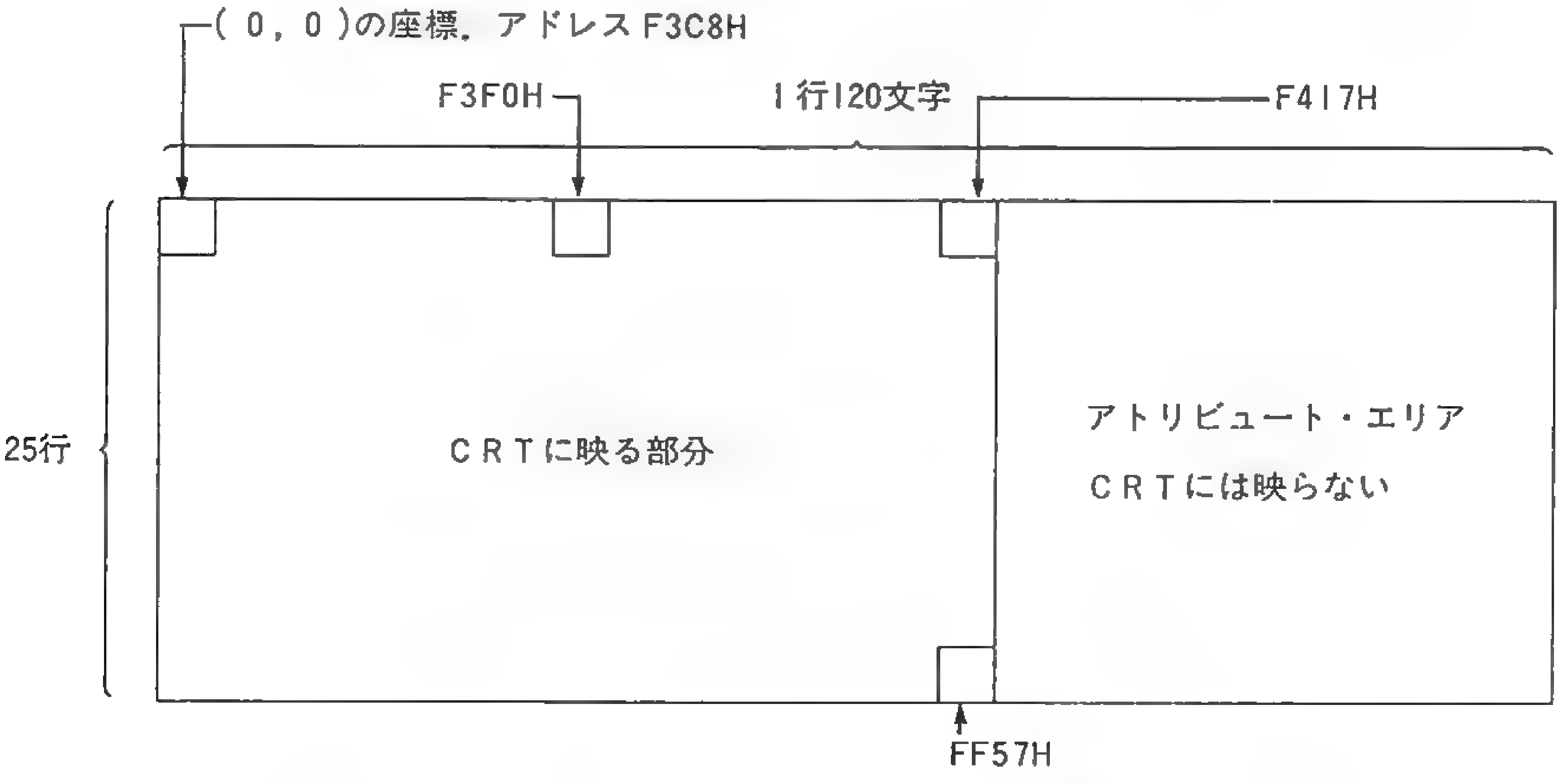


図5.8　PC-8801のテキストVRAMの構成

このプログラムのポイントは，キャラクタ座標の計算にあります．PC-8801のテキストVRAMの構成は図5．8のようになっており，CRTに映る部分は80文字ですが，1行につき40バイトのアトリビュート・エリアが付加されるため，あるキャラクタ座標の下の座標のメモリ・アドレスを計算するには120を加えるようにします．

5．6．1　ADD命令とADC命令(8ビット)

ADDは，その字が示す通り加算命令です．ここに出てくるADDは特に8ビットの加算を行うものですので，常に第1オペランドにはAレジスタが置かれ，第2オペランドに置かれたレジスタまたは数値との加算が行われます．

リスト10の例をみてみましょう．

```
ADD     A,120 (120=78H)
```

この命令の前では，すでにAレジスタにLの値(F0H)が入っているので，ADD A,120は，A＝F0H+78Hとなり，Aレジスタの値は168Hになります．

ところが，各レジスタは8ビットですから，このときのAレジスタの値は8ビットで表現できる最大値(FFH＝255)を超えています．実はFFHを超えたときにCYフラグは1になっていますから，この結果をHレジスタに反映させればよいのです．それがADC命令です．

この命令は，Aレジスタの内容と第2オペランド，それにCYフラグを加えて，結果をAレジスタに入れる命令です．リスト10では，0を加えているので，一見何も加えていないようにみえますが，実はCYフラグの結果をHレジスタの値に加えているのです．

```
ADC     A,0
```

このプログラムはBレジスタにセットした値（25…キャラクタ画面をWIDTH　80,25にした場合の縦の行数)が0になれば終了します．

プログラムを少々変更して，HLレジスタ・ペアに加える値を121や，119にすると斜線が引けます．

■ リストの入力について

リスト10のマシン語を入力する前に以下のBASICコマンドをダイレクトステートメントで実行してください．

```
WIDTH   80,25
```

ニーモニック	命令の動作	命令のサンプル	
		ニーモニック	マシン語
ADD A,n	Aレジスタの内容と8ビットデータnを足して，結果をAレジスタに入れる	ADD A,50H	C6 50
ADC A,n	Aレジスタの内容と8ビットデータnとCYフラグの値を足して，結果をAレジスタに入れる	ADC A,50H	CE 50

命令のまとめ4

リスト10

```
                        ;
                        ;    SAMPLE ADD 1
                        ;
F3F0                    VRAMC: EQU  0F3F0H     テキストVRAMのキャラクタ座標 (40, 0)
                        ;
                               ORG  0E000H
                        ;
E000 162A                      LD   D,'*'      Dに'*'のアスキー・コードをロードする
E002 21F0F3                    LD   HL,VRAMC   HLにテキストVRAMのキャラクタ座標(40.0)
                                               のアドレス
E005 0619                      LD   B,25       Bに繰り返し回数をセットする
                        ;
E007 72                 LOOP:  LD   (HL),D     HLで示されるメモリ(キャラクタ座標(0 , n))に'*'を描き込む
E008 7D                        LD   A,L
E009 C678                      ADD  A,120
E00B 6F                        LD   L,A         HL=HL+120
E00C 7C                        LD   A,H
E00D CE00                      ADC  A,0
E00F 67                        LD   H,A
E010 10F5                      DJNZ LOOP       もしB≠0だったらLOOPへ飛ぶ
                        ;
E012 FF                        RST  38H        モニタへ戻る
                        ;
E013                           END
```

5.7 キャラクタ・スクウェア

ここまで読み進まれた読者の方は，すでにかなりのマシン語の知識を獲得しています．そしてちょうどこの節を終了するあたりで，基本的なマシン語は習得できたことでしょう．

さて，皆さんも簡単なプログラムにも飽きたころだと思いますので，そろそろやや長めのプログラムを登場させようと思っています．

リスト11〜13は，リスト10を発展させてテキストVRAM上の最外周を〝*〟で囲むものです．

どのプログラムも，アルゴリズムは全く同じ手法を用いており，違うのはテキストVRAM上のアドレスの計算方法です．リスト中の，枠線で囲んだ部分をよく見較べてください．

5.7.1 枠を描く

基本的には，HLレジスタ・ペアにキャラクタ座標(0，0)のメモリ・アドレスをロードし，リスト10と同様に120ずつHLレジスタ・ペアを加算して，まず〝*〟を左端に上から下へ書き込みます．この処理が終わったとき，HLレジスタ・ペアの指し示す値はキャラクタ座標(0，25)となっており，画面外にはみ出してしまいます．そのためHLレジスタ・ペアの値から120を引き，キャラクタ座標を(0，24)に直します．次に，HLレジスタ・ペアをインクリメントしながら，〝*〟を最下行に左から右の順に描き込みます．この後，やはりHLレジスタ・ペアの示す値は画面からはみ出していますのでデクリメントを1回行うことによって，キャラクタ座標(79，24)の値にします．今度は120ずつ減算しながら〝*〟を右端に下から上の順で描き込みます．同様に，HLレジスタ・ペア

へ120を加えた後，これをデクリメントしながら〝＊〟を最上行に右から左の順に描き込みます（図5．9参照）．

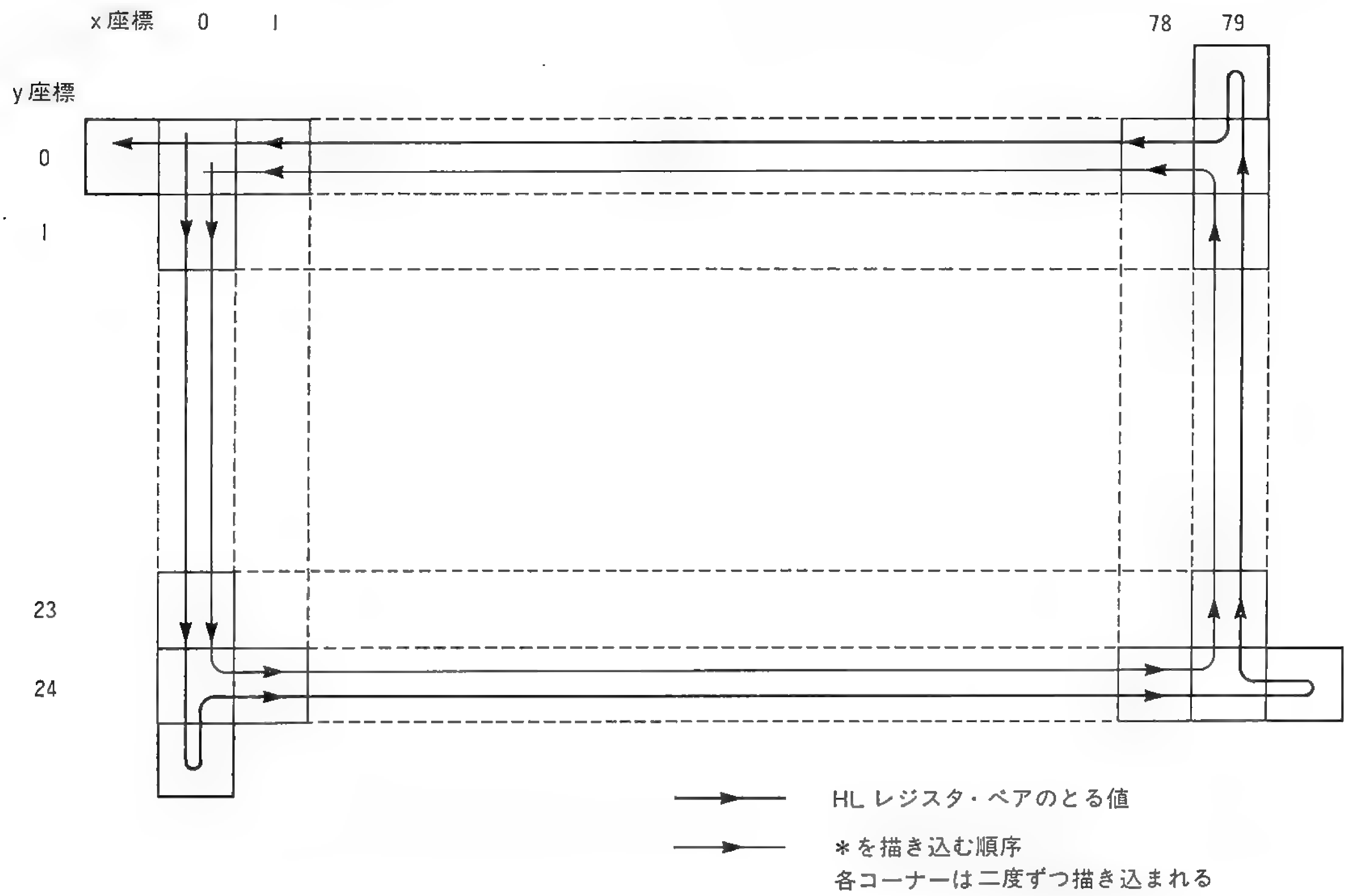

x座標
0
1
78
79
y座標
0
1
23
24
HL レジスタ・ペアのとる値
＊を描き込む順序
各コーナーは二度ずつ描き込まれる

図5.9　画面上のHLレジスタ・ペアのとる値と描く順序

5．7．2　SUB命令とSBC命令（８ビット）

どちらも８ビットの減算命令で，一般形は次のように表されます．

```
SUB     s
SBC     A,s
```

SUB sは，Aレジスタの内容からsで示される値を引き，結果をAレジスタに入れます．ニーモニックでは，SUB A,sとは表現しませんので注意してください．

SBC A,sは，Aレジスタの内容からsで示される値とCYフラグの値を引き，結果をAレジスタに入れます．

リスト11の例では，SBC命令は次のように使われています．０を引いているのは，リスト10のADCの例と同様に何も引いてないようにみえますが，CYフラグの値をAレジスタから引いています．

```
SBC     A,0
```

■リストの入力について

リスト11～13のマシン語を入力する前に，以下のBASICコマンドをダイレクトステートメントで実行してください．

```
WIDTH 80,25 : CONSOLE 1,23,0
```

リスト11

```
                  ;
                  ;;;;; EXAMPLE ADD & SUB  16 bit  [ DRAW BOX ] ;;;;;
                  ;
                  ;
F3C8              VRAM:  EQU  0F3C8H   テキストVRAMの先頭アドレス
0050              HSIZE: EQU  80       テキスト画面の横のサイズ
0019              VSIZE: EQU  25       テキスト画面の縦のサイズ
0078              LINE:  EQU  120
                  ;
                         ORG  0E000H
                  ;
E000 162A                LD   D,'*'    Dに'*'のアスキー・コードをロードする
E002 21C8F3              LD   HL,VRAM  HLにテキストVRAMの先頭アドレスをセットする
E005 0619                LD   B,VSIZE  繰り返り回数をセットする
                  ;
E007 72           LOOP1: LD   (HL),D   HLで示すメモリに'*'を描く      ┐
E008 7D                  LD   A,L      ┐                              │
E009 C678                ADD  A,LINE   │                              │
E00B 6F                  LD   L,A      │                              │
E00C 7C                  LD   A,H      ├HL=HL+120                     ├縦(左)に'*'を1列描く
E00D CE00                ADC  A,0      │                              │
E00F 67                  LD   H,A      ┘                              │
E010 10F5                DJNZ LOOP1    もしB≠0ならLOOP1へ飛ぶ         ┘
                  ;
E012 7D                  LD   A,L      ┐
E013 D678                SUB  LINE     │
E015 6F                  LD   L,A      │
E016 7C                  LD   A,H      ├HL=HL−120(キャラクタ座標(0,24)のアドレス)
E017 DE00                SBC  A,0      │
E019 67                  LD   H,A      │
E01A 0650                LD   B,HSIZE  ┘
                  ;
E01C 72           LOOP2: LD   (HL),D   HLで示すメモリに'*'を描く      ┐
E01D 23                  INC  HL       HL=HL−1                        ├横(下)に'*'を1列描く
E01E 10FC                DJNZ LOOP2    もしB≠0ならLOOP2へ飛ぶ         ┘
                  ;
E020 2B                  DEC  HL       HL=HL−1 (キャラクタ座標(79,24)のアドレス)
E021 0619                LD   B,VSIZE  繰り返し回数セードする
                  ;
E023 72           LOOP3: LD   (HL),D   HLで示すメモリに'*'を描く      ┐
E024 7D                  LD   A,L      ┐                              │
E025 D678                SUB  LINE     │                              │
E027 6F                  LD   L,A      │                              │
E028 7C                  LD   A,H      ├HL=HL−120                     ├縦(右)に'*'を1列描く
E029 DE00                SBC  A,0      │                              │
E02B 67                  LD   H,A      ┘                              │
E02C 10F5                DJNZ LOOP3    もしB≠0ならLOOP3へ飛ぶ         ┘
                  ;
```

```
E02E 7D              LD   A,L
E02F C678            ADD  A,LINE
E031 6F              LD   L,A          HL=HL+120 (キャラクタ座標(79, 0 )のアドレス)
E032 7C              LD   A,H
E033 CE00            ADC  A,0
E035 67              LD   H,A
E036 0650            LD   B,HSIZE      繰り返し回数をセットする
              ;
E038 72       LOOP4: LD   (HL),D       HLで示すメモリに'*'を描く
E039 2B              DEC  HL           HL=HL-1                   横(上)に'*'を1列描く
E03A 10FC            DJNZ LOOP4        もしB≠0ならLOOP4へ飛ぶ
              ;
E03C FF              RST  38H          モニタに戻る
              ;
E03D                 END
```

５．７．３　サブルーチンとCALL命令とRET命令

リスト11をみると枠線で囲んだ部分が４か所あり，このうち２か所ずつが全く同じ内容となっています．これをそれぞれ１か所にまとめてサブルーチン化したのがリスト12です．このサブルーチン化のために必要な命令が次の２つです．

```
CALL nn
RET
```

CALL命令はサブルーチンを呼ぶ命令で，RET命令はサブルーチンからメインルーチンへ戻る命令です．BASICにもサブルーチンを呼ぶ命令としてGOSUB～RETURNがありますが，これに相当するマシン語の命令がCALL／RET命令です．

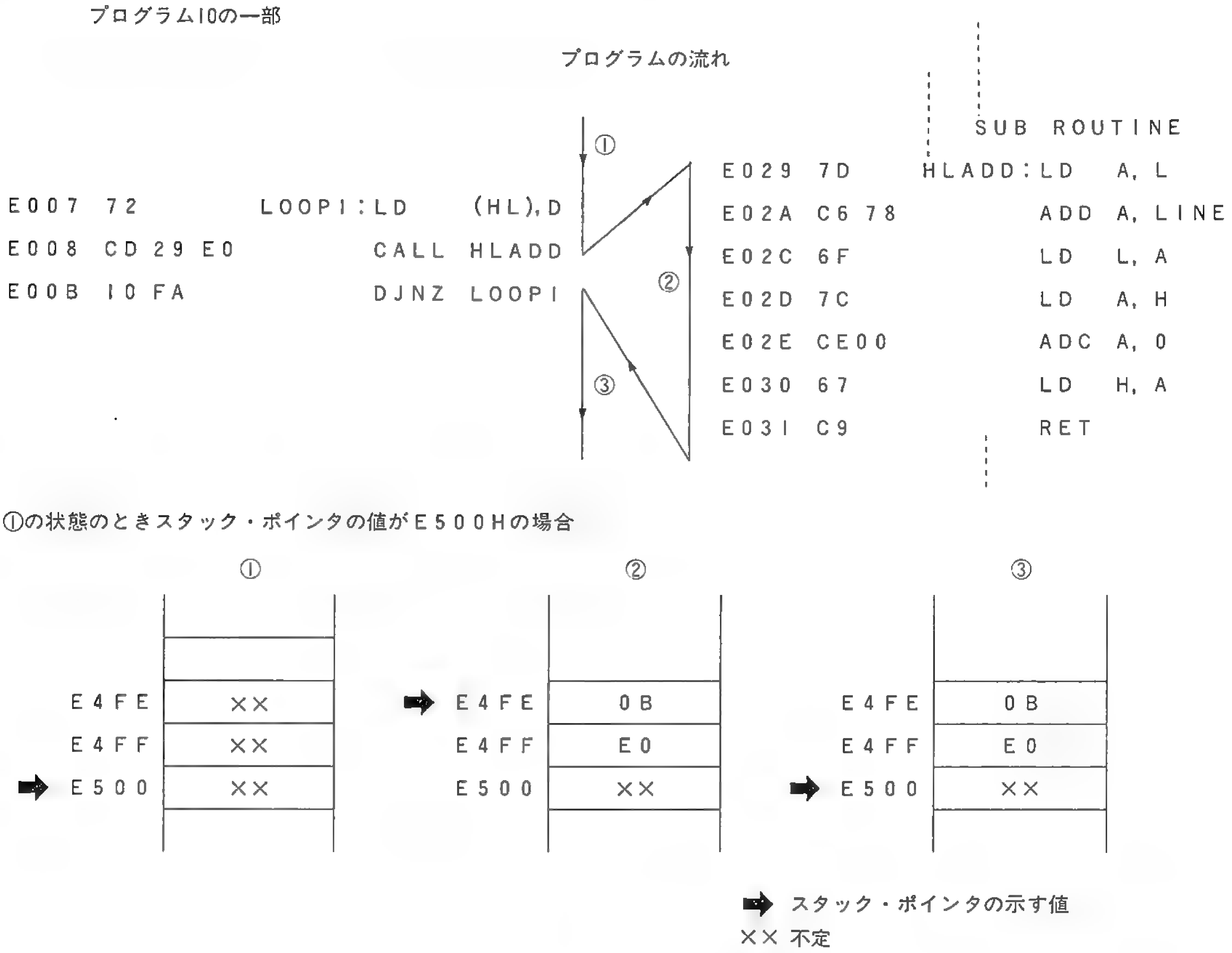
■ サブルーチンとスタック

さて，RET命令でサブルーチンからメインルーチンへ戻ることができますが，このときCPUはどんな動作を行うのでしょう．ここで重要なのが，スタック・ポインタとスタック領域の関係です．

リスト12のE008H番地から始まるCALL HLADDを例にとると，CPUはこの命令をフェッチした際，プログラム・カウンタに次の命令のあるアドレスE00BH番地をセットし，スタック・ポインタをデクリメントしてプログラム・カウンタの上位バイトE0Hをスタック・ポインタの示すアドレスのメモリ上にロードし，再びスタック・ポインタをデクリメントしてプログラム・カウンタの下位バイト0BHをスタック・ポインタの示すアドレスのメモリにロードします．この動作によって，スタック・ポインタの値は－2されたことになります．プログラム・カウンタには，この命令のオペランドであるHLADDのアドレスE029H番地がロードされ，サブルーチンHLADDへ分岐します．

さてCPUがサブルーチンの最後でRET命令をフェッチすると，スタック・ポインタの示すアドレスのメモリの内容0BHをプログラム・カウンタの下位バイトにロードしてスタック・ポインタを

インクリメントします．そして再びスタック・ポインタの示すアドレスのメモリの内容E0Hをプログラム・カウンタの上位バイトにロードしてスタック・ポインタをインクリメントします．この動作によってプログラム・カウンタには先程保存した値E00BHがロードされ，このサブルーチンを呼び出した次の命令に進むことができます．また，スタック・ポインタの値はサブルーチンを呼び出す前と同じ値になります(図５.10参照)．


プログラム10の一部

```
E007 72       LOOP1:LD   (HL),D
E008 CD 29 E0       CALL HLADD
E00B 10 FA          DJNZ LOOP1
```

```
                  SUB ROUTINE
E029 7D     HLADD:LD   A,L
E02A C6 78        ADD  A,LINE
E02C 6F           LD   L,A
E02D 7C           LD   A,H
E02E CE00         ADC  A,0
E030 67           LD   H,A
E031 C9           RET
```

①の状態のときスタック・ポインタの値がＥ５００Ｈの場合

	①	②	③
E4FE	××	➡ 0B	0B
E4FF	××	E0	E0
E500	➡ ××	××	➡ ××

➡ スタック・ポインタの示す値
×× 不定

図5.10 スタック領域のメモリの状態

■ RST

ここで，各プログラムの実行終了時に使用される命令RST 38Hについて説明しましょう．

RST 38Hの一般形は，RST pで表されます．この命令はCALL命令と同じ動作を行いますが，命令長が１バイトですので命令中に分岐先をもつことができず，すべてのアドレスへ分岐することはできません．この分岐先はpで表現されますが，これには以下の８種類があります．

00H , 08H , 10H , 18H , 20H , 28H , 30H , 38H

つまりRST 38Hは，命令長が違うことを除けばCALL 38Hと同じ働きをします．各分岐先のア

ドレスは通常ROMがあるところですから，ROMの内容によって役割が決まっており，ユーザの好きなことができるわけではありません．ですからここではRST 38Hだけを覚えればよいでしょう．他のRST命令を使用した場合にどうなるかは，『PC-Techknow8800Vol.1』(以降Techknowと略す)を参照してください．

ニーモニック	マシン語
RST 00H	C7
RST 08H	CF
RST 10H	D7
RST 18H	DF
RST 20H	E7
RST 28H	EF
RST 30H	F7
RST 38H	FF

図5.11 RST p

リスト12

```
                 ;
                 ;;;;; EXAMPLE CALL & RET  [ DRAW BOX ] ;;;;;
                 ;
                 ;
F3C8             VRAM:  EQU  0F3C8H   テキストVRAMの先頭アドレス
0050             HSIZE: EQU  80       テキスト画面の横のサイズ
0019             VSIZE: EQU  25       テキスト画面の縦のサイズ
0078             LINE:  EQU  120
                 ;
                        ORG  0E000H
                 ;
E000 162A               LD   D,'*'    Dに'*'のアスキー・コードをロードする
E002 21C8F3             LD   HL,VRAM  HLにテキストVRAMの先頭アドレスをセットする
E005 0619               LD   B,VSIZE  繰り返し回数をセットする
                 ;
E007 72          LOOP1: LD   (HL),D   HLで示すメモリに'*'を描く
E008 CD29E0             CALL HLADD    サブルーチンHLADDを呼ぶ      縦(左)に'*'を1列描く
E00B 10FA               DJNZ LOOP1    もしB≠0ならLOOP1飛ぶ
                 ;
E00D CD32E0             CALL HLSUB    サブルーチンHLSUBを呼ぶ
E010 0650               LD   B,HSIZE  繰り返し回数をセットする
                 ;
E012 72          LOOP2: LD   (HL),D   HLで示すメモリに'*'を描く
E013 23                 INC  HL       HL=HL+1                      横(下)に'*'を1列描く
E014 10FC               DJNZ LOOP2    もしB≠0ならLOOP2へ飛ぶ
                 ;
E016 2B                 DEC  HL       HL=HL−1 (キャラクタ座標(79, 0)のアドレス)
E017 0619               LD   B,VSIZE  繰り返し回数をセットする
                 ;
E019 72          LOOP3: LD   (HL),D   HLで示すメモリに'*'を描く
E01A CD32E0             CALL HLSUB    サブルーチンHLSUBを呼ぶ      縦(右)に'*'を1列描く
E01D 10FA               DJNZ LOOP3    もしB≠0ならLOOP3へ飛ぶ
```

```
                    ;
E01F CD29E0                 CALL HLADD     サブルーチンHLADDを呼ぶ
E022 0650                   LD   B,HSIZE   繰り返し回数をセットする
                    ;
E024 72             LOOP4:  LD   (HL),D    HLで示すメモリに'*'を描く  ┐
E025 2B                     DEC  HL        HL=HL−1                    ├横(上)に'*'を1列描く
E026 10FC                   DJNZ LOOP4     もしB≠0ならLOOP4へ飛ぶ      ┘
                    ;
E028 FF                     RST  38H       モニタへ戻る
                    ;
                    ;  SUB ROUTINE
                    ;
E029 7D             HLADD:  LD   A,L       ┐
E02A C678                   ADD  A,LINE    │
E02C 6F                     LD   L,A       │
E02D 7C                     LD   A,H       ├HL=HL+120
E02E CE00                   ADC  A,0       │
E030 67                     LD   H,A       ┘
E031 C9                     RET
                    ;
E032 7D             HLSUB:  LD   A,L       ┐
E033 D678                   SUB  LINE      │
E035 6F                     LD   L,A       │
E036 7C                     LD   A,H       ├HL=HL−120
E037 DE00                   SBC  A,0       │
E039 67                     LD   H,A       ┘
E03A C9                     RET
                    ;
E03B                        END
```

5.7.4　16ビット加／減算命令とCYフラグの操作

リスト11と12は，テキストVRAM上のアドレス計算に8ビットの加／減算命令を使用していたために，HLレジスタ・ペアに120を加えるのに非常にめんどうな方法をとりました．この8ビットで計算を行う部分に16ビット加／減算命令を使用したのがリスト13です．

ただ，16ビット減算命令にはSBC HL,ssしかないために，CYフラグの状態によって結果に影響が出てしまいます．そこでCYフラグをリセットするためにSCFとCCFを使用しています．
ただし，CYフラグをリセットする方法には別の命令を使用するのが一般的です．ここでは登場しませんでしたが，皆さんご自身で考えてください．

リスト13

```
                    ;
                    ;;;;; EXAMPLE ADD & SUB 1 [ DRAW BOX ] ;;;;;
                    ;
                    ;
F3C8                VRAM:   EQU  0F3C8H    テキストVRAMの先頭アドレス
0050                HSIZE:  EQU  80        テキスト画面の縦のサイズ
0019                VSIZE:  EQU  25        テキスト画面の横のサイズ
0078                LINE:   EQU  120
                    ;
                            ORG  0E000H
                    ;
E000 0E2A                   LD   C,'*'     Cに'*'のアスキー・コードをロードする
E002 117800                 LD   DE,LINE   DE=120
```

```
E005 21C8F3            LD   HL,VRAM  HLにテキストVRAMの先頭アドレスをセットする
E008 0619              LD   B,VSIZE  繰り返し回数をセットする
                ;
E00A 71         LOOP1: LD   (HL),C   HLの示すメモリへ'*'を描き込む       ┐
E00B 19                ADD  HL,DE    HL=HL+120                          ├縦(左)に'*'を1列描く
E00C 10FC              DJNZ LOOP1    もしB≠0だったらLOOP1へ飛ぶ          ┘
                ;
E00E 37                SCF           ┐CYフラグを0にする
E00F 3F                CCF           ┘
E010 ED52              SBC  HL,DE    HL=HL−120(キャラクタ座標(0,24)のアドレス)
E012 0650              LD   B,HSIZE  繰り返し回数をセットする
                ;
E014 71         LOOP2: LD   (HL),C   HLの示すメモリへ'*'を描き込む       ┐
E015 23                INC  HL       HL=HL+1                            ├横(下)に'*'を1列描く
E016 10FC              DJNZ LOOP2    もしB≠0だったらLOOP2へ飛ぶ          ┘
                ;
E018 2B                DEC  HL       HL=HL−1 (キャラクタ座標(79,24)のアドレス)
E019 0619              LD   B,VSIZE  繰り返し回数をセットする
                ;
E01B 71         LOOP3: LD   (HL),C   HLの示すメモリに'*'を描き込む       ┐
E01C 37                SCF           ┐CYフラグを0にする                 │
E01D 3F                CCF           ┘                                  ├縦(右)に'*'を1列描く
E01E ED52              SBC  HL,DE    HL=HL−120                          │
E020 10F9              DJNZ LOOP3    もしB≠0だったらLOOP3へ飛ぶ          ┘
                ;
E022 19                ADD  HL,DE    HL=HL+120(キャラクタ座標(70,0)のアドレス)
E023 0650              LD   B,HSIZE  繰り返し回数をセットする
                ;
E025 71         LOOP4: LD   (HL),C   HLの示すメモリに'*'を描く           ┐
E026 2B                DEC  HL                                          ├横(上)に'*'を1列描く
E027 10FC              DJNZ LOOP4    もしB≠0だったらLOOP4へ飛ぶ          ┘
                ;
E029 FF                RST  38H      モニタへ戻る
                ;
E02A                   END
```

ニーモニック	命令の動作	命令のサンプル	
		ニーモニック	マシン語
SUB s	Aレジスタの内容から8ビットデータsの内容を引いて，結果をAレジスタに入れる	SUB B	90
SBC A, s	Aレジスタの内容から8ビットデータsの内容とCYフラグの値を引き，結果をAレジスタに入れる	SBC A, B	98
CALL nn	nnで指定されるアドレスのサブルーチンを呼び出す	CALL FF00H	CD 00FF
RET	無条件でメインルーチンへ戻る	RET	C9
ADD HL, ss	HLレジスタ・ペアの内容とssレジスタ・ペアの内容を足して結果をHLレジスタ・ペアに入れる	ADD HL, BC	09
SBC HL, ss	HLレジスタ・ペアの内容からssレジスタ・ペアの内容とCYフラグの値を引いて結果をHLレジスタ・ペアに入れる	SBC HL, BC	ED 42
ADC HL, ss	HLレジスタ・ペアの内容とssレジスタ・ペアの内容とCYフラグの値を足して結果をHLレジスタ・ペアに入れる	ADC HL, BC	ED 4A
SCF	CYフラグをセットする	SCF	37
CCF	CYフラグの状態を反転する	CCF	3F

命令のまとめ5

5.7.5 プログラムバリエーション

これらのプログラムは，リスト10を発展させたためにループ回数が多く，HLレジスタ・ペアの指し示す値がいちいち画面外にはみ出てしまい，それを補正しています．各々ループの回数を1回ずつ減らせば，この補正の必要はなくなり，プログラム自体もずっとすっきりしますので，ぜひ試してみてください．

5.8 G-VRAMの接続を調べる

N_{88}-BASICは，112Kバイトものを巧みに切り換えて使用しており，この切り換えの制御に使われているのがI/Oポートの一部です．I/Oポートはメモリと異なり，出力した(書き込んだ)内容と入力した(読み込んだ)内容は必ずしも一致しません．というのは，I/Oポートの中には出力専用のポートと入力専用のポート，そして入/出力両方に使用できるものがあり，入力専用ポートと出力専用ポートを同じポート・アドレスに割り当てることがあるからです．リスト14は，グラフィックVRAM(以下G-VRAM)を切り換えて，現在どのG-VRAMまたはメインRAMが選択されているかを調べ，その結果を表示するプログラムです．

リスト14

```
                 ;
                 ;    SAMPLE   IN   BIT
                 ;
F3C8             VRAM:   EQU   0F3C8H
                 ;
005C             RAMSEL:EQU    5CH           G-VRAMのステータス・ポートのアドレス
005C             GVRAM0:EQU    5CH           G-VRAM 0の選択ポートのアドレス
005D             GVRAM1:EQU    5DH           G-VRAM 1の選択ポートのアドレス
005E             GVRAM2:EQU    5EH           G-VRAM 2の選択ポートのアドレス
005F             MRAM:   EQU   5FH           メインRAM選択ポートのアドレス
                 ;
                         ORG   0BC00H
                 ;
BC00 F3                  DI                  割込み禁止
BC01 D35E                OUT   (GVRAM2),A    G-VRAMのステータスをコピーする
BC03 DB5C                IN    A,(RAMSEL)    G-VRAM 2を選択する
                 ;
BC05 CB47                BIT   0,A           }選択されたのがG-VRAM 0かを判断し、違っていればNEXT 1
BC07 2805                JR    Z,NEXT1       }へ飛ぶ
BC09 2133BC              LD    HL,MES0       HLにMES 0の先頭アドレスをセットする
BC0C 1815                JR    OUTMES        OUTMESへ
                 ;
BC0E CB4F        NEXT1:  BIT   1,A           }選択されたのがG-VRAM 1かを判断し,違っていればNEXT 2
BC10 2805                JR    Z,NEXT2       }へ飛ぶ
BC12 2150BC              LD    HL,MES1       HLにMES 1の先頭アドレスをセットする
BC15 180C                JR    OUTMES        OUTMESへ飛ぶ
                 ;
BC17 CB57        NEXT2:  BIT   2,A           }選択されたのがG-VRAM 2かを判断し,レジスタAの第2ビット
BC19 2805                JR    Z,NEXT3       }が0ならばNEXT 3へ飛ぶ.違っていればNEXT 3へ飛ぶ
BC1B 216CBC              LD    HL,MES2       HLにMES2の先頭アドレスをセットする
BC1E 1803                JR    OUTMES        OUTMESへ飛ぶ
                 ;
BC20 218ABC      NEXT3:  LD    HL,MES3       HLにMES3の先頭アドレスをセットする（メインRAMが選択さ
                                             れたと判断）
                 ;
BC23 D35F        OUTMES:OUT   (MRAM),A      メインRAMを選択する
BC25 FB                 EI                  割込み許可
BC26 1140F4             LD    DE,VRAM+120   DEにキャラクタ座標（0，1）のアドレスをセットする
BC29 7E          LOOP:  LD    A,(HL)        AにHLで示されるメモリの内容(メッセージ)をロードする
BC2A B7                 OR    A            }Aが0つまり表示するデータが終りならESCAPEへ飛ぶ
BC2B 2805               JR    Z,ESCAPE     }
BC2D 12                 LD    (DE),A        DEの示すメモリへAの内容(メッセージ)を出力する
BC2E 13                 INC   DE            DE=DE+1
BC2F 23                 INC   HL            HL=HL+1
BC30 18F7               JR    LOOP          LOOPへ飛ぶ
                 ;
BC32 FF          ESCAPE:RST   38H           モニタへ戻る
                 ;
BC33 53454C45 MES0:  DB    'SELECT GRAPHIC VRAM 0 (BLUE)',0
BC37 43542047
BC3B 52415048
BC3F 49432056
BC43 52414D20
BC47 30202842
BC4B 4C554529
BC4F 00
BC50 53454C45 MES1:  DB    'SELECT GRAPHIC VRAM 1 (RED)',0
BC54 43542047
BC58 52415048
BC5C 49432056
BC60 52414D20
BC64 31202852
BC68 45442900                                             出力メッセージ
BC6C 53454C45 MES2:  DB    'SELECT GRAPHIC VRAM 2 (GREEN)',0
BC70 43542047
BC74 52415048
BC78 49432056
BC7C 52414D20
BC80 32202847
BC84 5245454E
BC88 2900
BC8A 53454C45 MES3:  DB    'SELECT MAIN MEMORY',0
BC8E 4354204D
BC92 41494E20
BC96 4D454D4F
BC9A 525900
              ;
BC9D                 END
```

プログラム中に使用されているポートは，入力ポートとして5CH，出力ポートとして5CH，5DH，5EH，5FHの4つを用いています．

5.8.1　ポートのはなし

まず出力ポートについて説明しましょう．

5CH，5DH，5EH，5FHは，それぞれ書き込み動作によってG-VRAMおよびメインRAMを選択します．出力するデータはどのようなものでもよく，とにかく書き込めばよいのです(図5.12参照)．

ポート	
5CH	G-VRAM 0（青）を選択する
5DH	G-VRAM 1（赤）を選択する
5EH	G-VRAM 2（緑）を選択する
5FH	メインRAMを選択する

例えば5EHに書き込みを行うとメモリマップは以下のようになります．

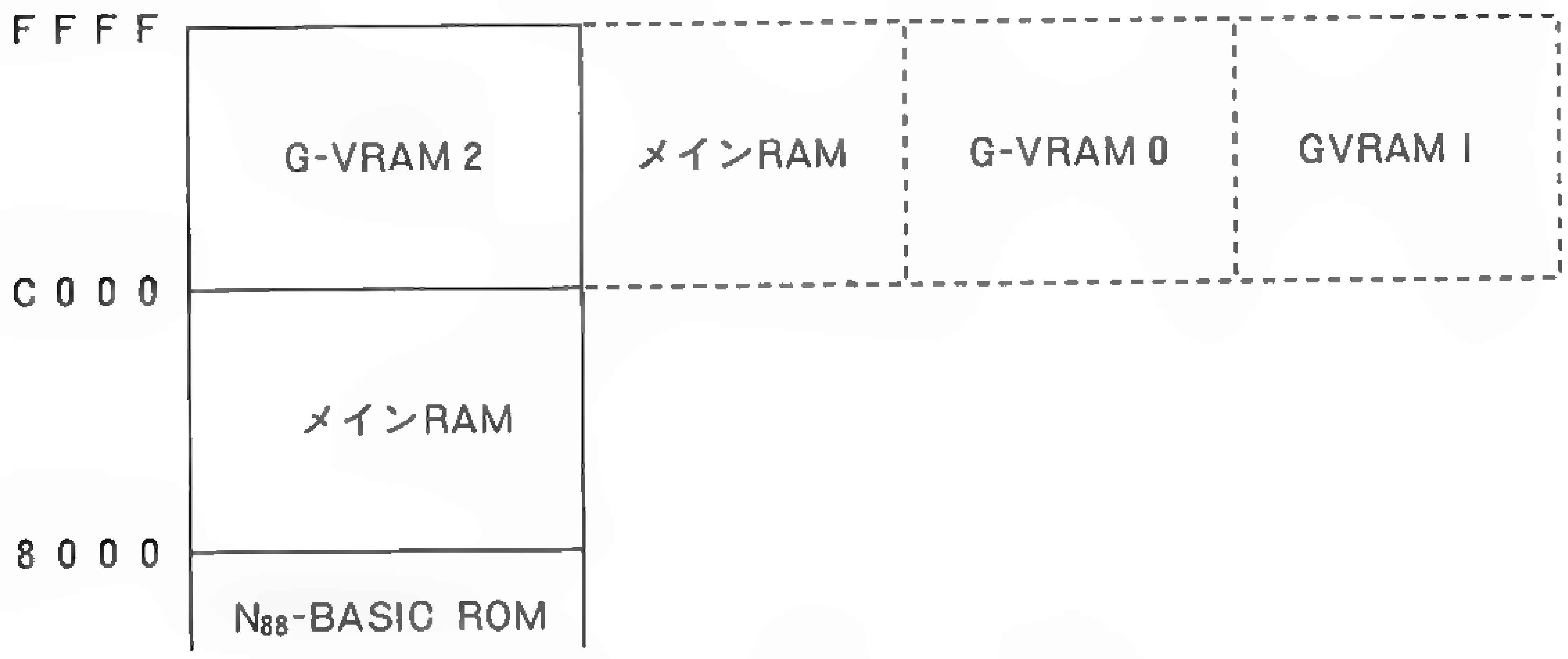


図5.12　G-VRAMの選択

また入力ポート5CHは，現在接続されているG-VRAMを調べるポートです．入力データのビット0～2が，それぞれG-VRAMの0～2に対応しており，上位5ビットは意味をもちません．

ビット0が1の場合：　G-VRAM0が選択されている
ビット1が1の場合：　G-VRAM1が選択されている
ビット2が1の場合：　G-VRAM2が選択されている
ビット0～2が0の場合：メインRAMが選択されている

5.8.2　IN／OUT命令とBIT命令

それぞれのI／Oポートは上で述べたような役割をもっていますが，I／Oポートの働きをマシン語でコントロールするのがIN／OUT命令です．実際にリスト14の中でどのように使われているかをみてみましょう．

BC01H番地から始まるOUT　(GVRAM2),Aをみると，第1オペランドは(GVRAM2)で，これはEQU 5EHでポートは5EHが指定されたことになります．そしてこの命令では第2オペランドのAレジスタの内容をポート5EHに出力するという働きをします．このときのAレジスタの内容は特に指定されていませんが，これはOUT命令によって特定のポートに何かが出力されたという動作があればよいからです．このようにOUT命令は一般にレジスタからポートにデータを出力する働きがあります．

次に，IN命令の例をみてみましょう．今のOUT命令のすぐ後に出ています．IN命令の働きはOUTの逆，つまり特定のポートの内容をレジスタに送るもので，この例ではRAMSELで指定されたポート5CHの内容をAレジスタに送っているのです．

さて，IN命令と組み合わせてよく使われる命令にBIT命令があります．リスト14でもBC05H番地からBIT 0,AとしてIN命令の後に使われています．この命令は，レジスタやメモリの特定のビットがセットあるいはリセットされているかを調べるもので，この命令の実行結果によってZフラグが動作するために，条件判断の材料として利用されます．ここではAレジスタのビット０の状態を調べ相対分岐の条件として利用しています．

5.8.3　OR命令

BC2AH番地でOR Aが使用されていますが，これは，表示するデータの終了を判定するために，Aレジスタの値が０かどうかを調べるためです．ORの真理値表を思い出してください．AとBの値が等しい場合，Yの値はA(またはB)の値と等しくなります．OR命令はZフラグに影響を与えますから，OR Aを行うとAレジスタの内容が０かどうかを判定できます．

ニーモニック	命令の動作	命令のサンプル	
		ニーモニック	マシン語
OUT (n),A	nで示されるポートにAレジスタの内容を出力する	OUT (FEH),A	D3 FE
IN A,(n)	nで示されるポートの状態をAレジスタにロードする	IN A,(5CH)	DB 5C
DI	割込みを禁止する	DI	F3
EI	割込みを許可する	EI	FB
BIT b,r	rレジスタのbビットがセットされているか否かを調べる 指定されたビットが1ならばZフラグがリセットされ,0ならばZフラグがセットされる	BIT 0,A	CB 47
JR Z,e	Zフラグがセットされていれば PCの内容にe-2を足した値で示すアドレスへジャンプする	JR Z,e	28 e-2
JR e	無条件に,PCの内容にe-2を足した値で示すアドレスへジャンプする	JR e	18 e-2
OR r	Aレジスタの内容とrレジスタの内容の論理和(OR)をとり結果をAレジスタに入れる	OR A	B7

命令のまとめ６

5.8.4　DI命令とEI命令

さて，最後になりましたが，リスト14ではDIとEIという新しい命令も使われています．DIは割込み禁止，EIは割込み許可と呼ばれる命令なのですが，今のところは割込みということは考えないで，ここではRAMバンクを選択する際に必ず実行しなければならない命令がDIであり，DIを行った場合にモニタのコマンド・レベルに戻る前に必ず実行する命令がEIであると考えておいてください．

5.8.5　G-VRAMの選択

それではリスト14のBC01H番地からのプログラムの意味を追っていきます．まず，OUT (G-VRAM2),Aを行うことによってG-VRAM2バンク(緑)をCPUのアドレス空間に割り当てます．そ

の後，ポート5CHから読み込んだデータのビット 0 ～ 2 を調べることによって現在選択されているRAMを判断し，画面にメッセージを表示します．プログラムが正常に動作すれば画面には以下のメッセージが出力されます．このプログラムの中でBC02H番地の5EHという値を5CH～5FHに変えて実行してみてください．実行後結果のメッセージが変わります．

```
SELECT GRAPHIC VRAM 2(GREEN)
```

5.8.6 メモリ・ロケーション

リスト14の例は，今までのプログラムとは異なりORG 0BC00Hとなっています．図 5 .13をみていただければわかるように，もし今までのプログラムのようにスタート・アドレスがE000H番地であったとすれば，メインRAM(MRAM)を切り離したとたんにプログラムがCPUから失われてしまい，暴走を起こすことになります．そのため，スタート・アドレスはG-VRAMを切り換えても影響の出ないアドレス内にプログラムが納まるように選んであります．

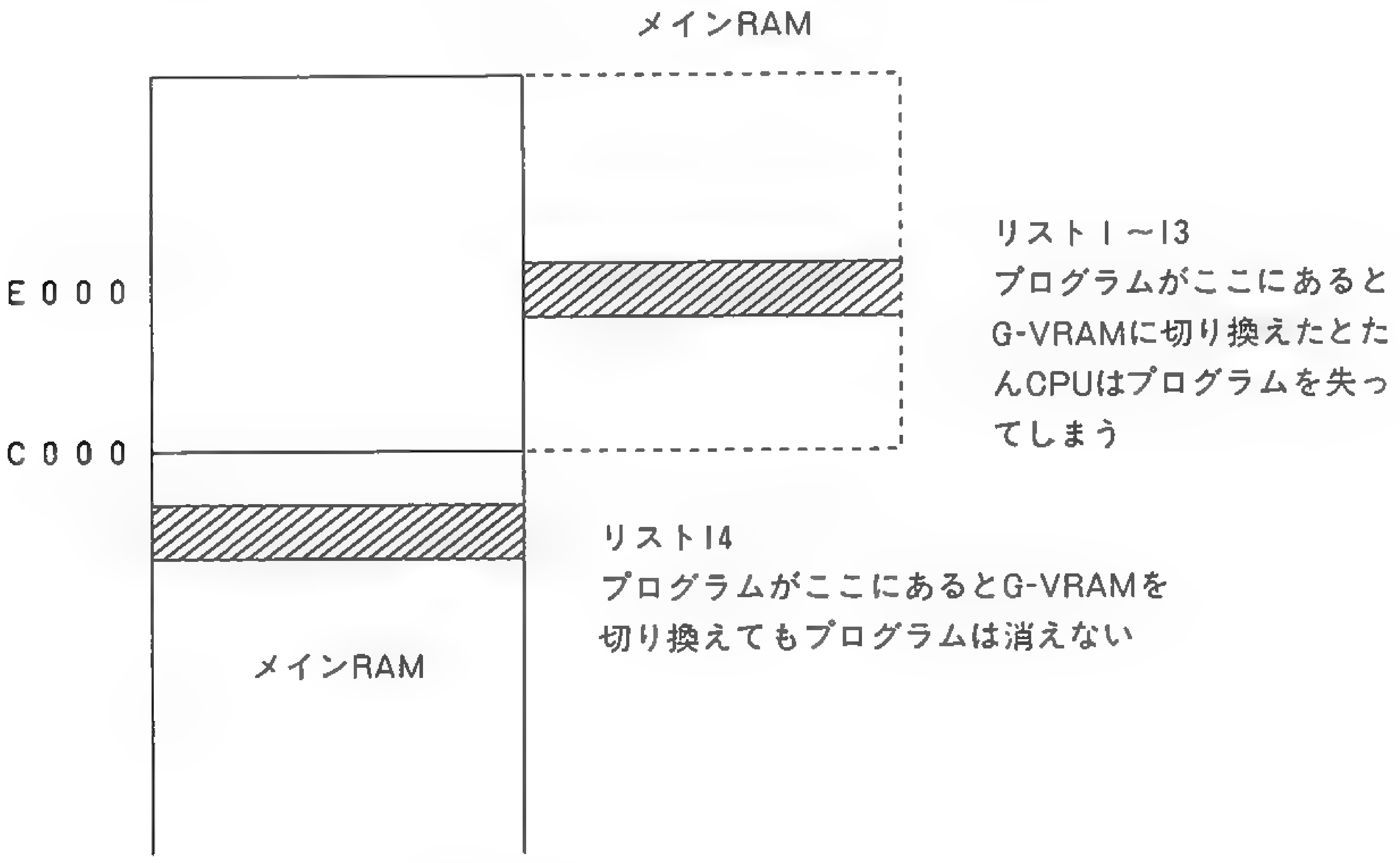

メインRAM
E000
C000
リスト 1 ～13
プログラムがここにあると
G-VRAMに切り換えたとたんCPUはプログラムを失ってしまう
リスト14
プログラムがここにあるとG-VRAMを切り換えてもプログラムは消えない
メインRAM

図5.13 メモリ・ロケーション

5.9 グラフィック画面のイニシャライズ

このプログラムは，BASICコマンドでCLS 2を行うよりはるかに高速です．ただし，白黒モードがSCREEN文で選択されている場合はアクティブ・ページについていっさい関知しませんので，BASICで何かを書き込む場合，改めてSCREEN 0を行う必要があります．

リスト15は，以下のことをマシン語で行います．

高速書き込みスイッチ　OFF
グラフィック・マスクスイッチ　OFF
カラー・パレットのイニシャライズ

カラーモードの選択

グラフィック画面のクリア(VIEWの値に関係なく全画面)

リスト15

```
                        ;
                        ;  INITIALIZE FOR GRAPHIC VRAM
                        ;
                        ;
C000                    GVRAM:  EQU   0C000H          G-VRAMの先頭アドレス
E6C1                    SSYC1:  EQU   0E6C1H          I/Oポート40Hのデータが保存してあるワークエリア内のアドレス
E6C2                    SCRTC:  EQU   0E6C2H          I/Oポート31Hのデータが保存してあるワークエリア内のアドレス
                        ;
0031                    CONT2:  EQU   31H             システムコントロール・ポートのアドレス
0040                    STROB:  EQU   40H             ストローブ・ポートのアドレス
0053                    VRAMAC:EQU    53H             表示するVRAMを決定するアドレス
0054                    PALETT:EQU    54H             カラーパレットコントロール・ポートの先頭アドレス
005C                    GVRAM0:EQU    5CH             G-VRAM選択ポートの先頭アドレス
005F                    MRAM:   EQU   5FH             メインRAMの選択ポートのアドレス
                        ;
                                ORG   0BC00H
                        ;
BC00 3AC1E6                     LD    A,(SSYC1)       AにSSYC1のデータ(ポート40Hに出力したデータ)をロードする
BC03 CBA7                       RES   4,A             }G-VRAMの描き込む速さをノーマルにセットする
BC05 D340                       OUT   (STROB),A       }
BC07 32C1E6                     LD    (SSYC1),A       ポート40Hに出力したデータを保存する
BC0A 3AC2E6                     LD    A,(SCRTC)       AにSCRTCのデータ(ポート31Hに出力したデータ)をロードする
BC0D CBDF                       SET   3,A             }
BC0F CBE7                       SET   4,A             }グラフィック画面を表示し，カラー・モードをセットする
BC11 D331                       OUT   (CONT2),A       }
BC13 32C2E6                     LD    (SCRTC),A       ポート31Hに出力したデータを保存する
BC16 AF                         XOR   A               }すべてのVRAMを表示する
BC17 D353                       OUT   (VRAMAC),A      }
                        ;
BC19 0E54                       LD    C,PALETT        }
BC1B 0608                       LD    B,8             }
BC1D ED79               INIT1:  OUT   (C),A           }カラー・パレットのイニシャライズ
BC1F 3C                         INC   A               }
BC20 0C                         INC   C               }
BC21 10FA                       DJNZ  INIT1           }
                        ;
BC23 F3                         DI                    割込み禁止
BC24 3E5C                       LD    A,GVRAM0                                     }
BC26 4F                 INIT2:  LD    C,A                                          }
BC27 ED79                       OUT   (C),A                                        }
BC29 2100C0                     LD    HL,GVRAM        }                            }
BC2C 1101C0                     LD    DE,GVRAM+1      }                            }
BC2F 017F3E                     LD    BC,15999        }1画面分のG-VRAMをクリア      }G-VRAMをクリアする
BC32 3600                       LD    (HL),0          }                            }
BC34 EDB0                       LDIR                  }                            }
BC36 3C                         INC   A                                            }
BC37 FE5F                       CP    MRAM            3画面分クリアしたかを判定     }
BC39 20EB                       JR    NZ,INIT2                                     }
                        ;
BC3B D35F                       OUT   (MRAM),A        メインRAMを選択する
BC3D FB                         EI                    割込み許可
                        ;
BC3E FF                         RST   38H             モニタに戻る
                        ;
BC3F                            END
```

5.9.1　ポートの役割

さて，リスト15で新たに使用した出力ポートについて説明しましょう．

▶31H

リスト15ではCONT2というラベルに定義して使用しています．主にメモリとCRTモードをコントロールするポートで，N_{88}-BASICではこのポートに出力したデータをワークエリアのE6C2H番地に記憶していますので，この値をロードして，ビット3と4をセットしてポートに出力し，変更したデータをE6C2H番地に戻しています．このようにポートの状態を変えた場合に，もしそのデータをN_{88}-BASICが記憶しているならば，必ず変更したデータをその記憶場所に戻してやることが必要です．

さて，ビット3は何を行うかといいますと，このビットが0のときはグラフィック画面を表示せず，1のとき表示します．ここではグラフィック画面を表示するために1としています．ビット4は，グラフィック画面をカラーにするか白黒にするかを決めるビットで，0のとき白黒，1のときカラーとなります．ここでは1にすることによってカラー・モードにしています．他のビットについては，触れない方が無難だと思いますが，興味のある方は『Techknow』を参照してください．

▶40H

リスト15ではSTROBというラベルに定義して使用しています．このポートは，各種コントロール信号を出力するポートで，ここに出力したデータはメモリ・アドレスE6C1H番地に記憶してあります．ここではE6C1H番地のメモリの内容をロードして，ビット4のリセットを行い，いったんポートに出力して，そのデータをメモリに戻しています．ビット4をリセットしたのは，高速書き込みスイッチをOFFにするためです．もし高速書き込みスイッチをONにしたい場合は，このビットをセットすればよいのです．

▶53H

画面の重ね合わせの制御を行うポートです．ビット0～3の下位4ビットのみが意味をもちます．

ビット0	テキスト画面	各ビットが0のとき表示される
ビット1	G-VRAM0	
ビット2	G-VRAM1	
ビット3	G-VRAM2	

リスト15では，このポートに0を書き込むことによって全画面を表示しています．

▶54H～5BH

カラー・パレットに対応する色を決定するポートです．それぞれビット0～2の下位3ビットのみが意味をもちます．

ビット0	青
ビット1	赤
ビット2	緑

54H	カラー・パレット0の制御
55H	カラー・パレット1の制御
56H	カラー・パレット2の制御
57H	カラー・パレット3の制御
58H	カラー・パレット4の制御
59H	カラー・パレット5の制御
5AH	カラー・パレット6の制御
5BH	カラー・パレット7の制御

リスト15では，ループを組むことによってカラー・パレット0に黒，1に青，2に赤…となるようにイニシャライズを行っています．

5.9.2　XOR命令

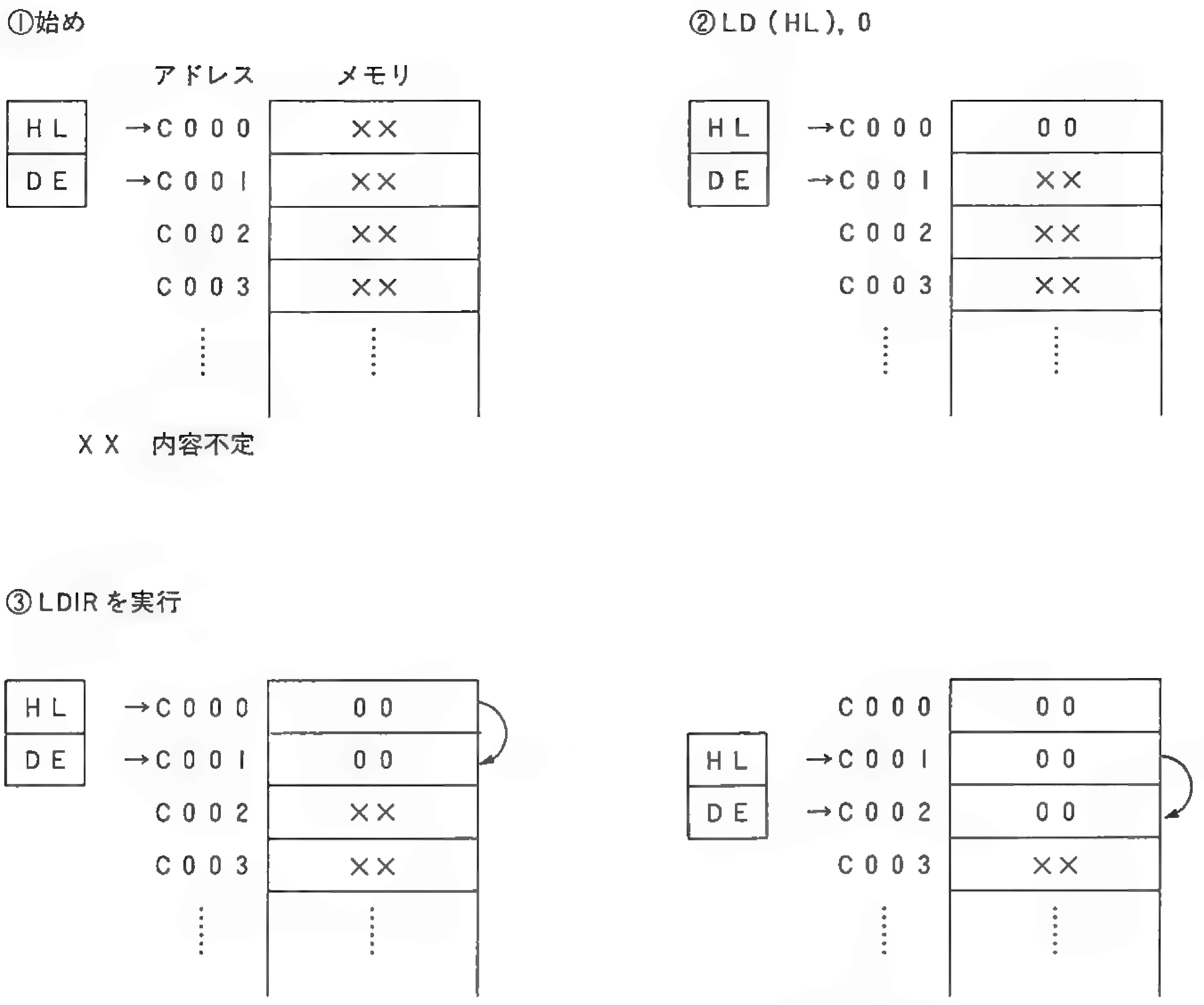
リスト15での，XORはAレジスタとAレジスタの値どうしでXORをとっています．これは，XORの真理値表を思い出していただけばわかるようにAとBの値が等しいときは$Y=A\forall B=0$となり，これによって，Aレジスタの内容を0にしています．

5.9.3　LDIR命令

LDIRはHLレジスタ・ペアに転送元のアドレスをセットし，BCレジスタ・ペアに転送するバイト数をセットし，さらにDEレジスタ・ペアに転送先のアドレスをセットすれば，たった1つの命令でメモリの内容が転送されます．なお，この命令はブロック転送命令と呼ばれます．

リスト15においては，この命令をうまく利用することによってグラフィック画面のクリアを行っています(図5.14参照)．

ブロック転送命令は，この他にLDI／LDD／LDDRといった命令があります．これらの命令は連続したメモリ・アドレスに格納されている大量のデータを別のメモリ領域に転送する場合に便利です．ここではLDIR命令しか使われていませんが，他の命令についてはインストラクション表を参照してください．


①始め

②LD (HL), 0

③LDIRを実行

こうしてメモリに同じ内容(00)が書き込まれる

図5.14 LDIRによるG-VRAMのクリア

5.9.4 CP命令

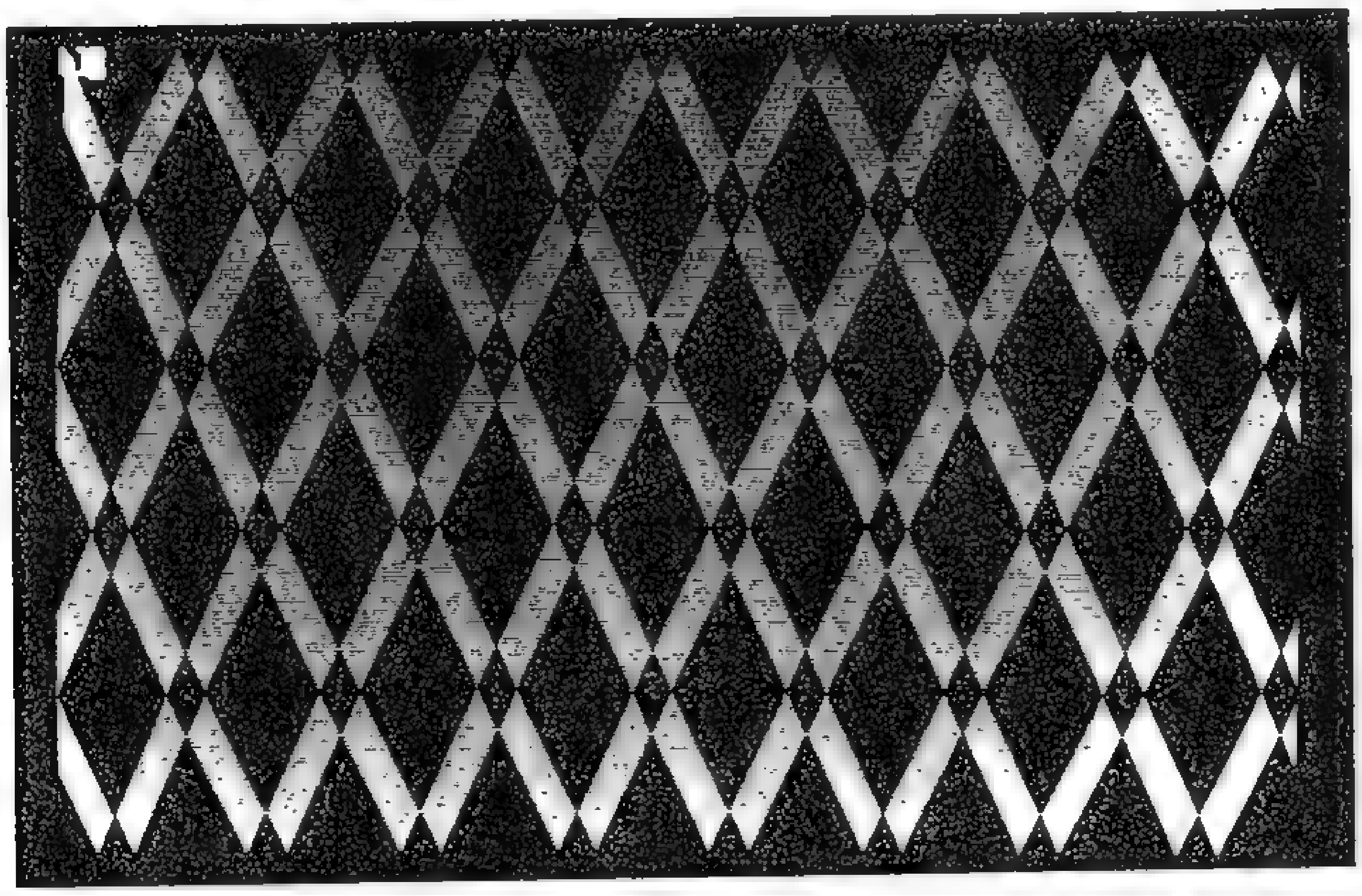
CP命令は一般にCP sと表され，Aレジスタの内容とsの値とを比較する命令です．重要なのは比較をするという点で，この比較とは，SUB sを行うことです．ただし減算は行われますがAレジスタの内容は変化せず，フラグのみが変化します．つまり比較命令とは，減算を行ってその結果をフラグに反映させる命令なのです．

リスト15においては，Aレジスタの値を調べてループ終了条件に使用しました．この部分について説明を加えると，Aレジスタにはまず G-VRAM0を選択するポートの値をロードします．これをCレジスタにロードして，OUT (C),Aを行うことにより，この画面のクリアを行います．LDIRの終了後，Aレジスタをインクリメントして CP MRAMでAレジスタの内容を5Fと比較し，違っていればBC26H番地に分岐して次にG-VRAM1を選択して，これをクリアします．この部分は，PC-8801のハードウェアの出力ポートが，5CH～5FHの順にG-VRAM0～2およびメインRAMを選択するようにできているのを利用したものです．

ニーモニック	命令の動作	命令のサンプル	
		ニーモニック	マシン語
RES b,r	rレジスタのbビットをリセットする	RES 7,E	CB BB
SET b,r	RES b,rの逆の命令にあたり，rレジスタのbビットをセットする	SET 0,A	CB C7
XOR r	Aレジスタの内容とrレジスタの内容の排他的論理和(XOR)をとり結果をAレジスタに入れる	XOR A	AF
OUT (C),r	Cレジスタの内容で示されるポートにrレジスタの内容を出力する	OUT (C),A	ED 79
LD (HL),n	HLレジスタ･ペアで示されるアドレスのメモリに，8ビットデータnをロードする	LD (HL),05H	36 05
LDIR	HLレジスタ・ペアの示すアドレスのメモリ内容をDEレジスタ・ペアの示すアドレスのメモリにロードし，HL，DEレジスタ・ペアをインクリメント，BCレジスタ・ペアをデクリメントする．この動作をBCレジスタ・ペアが0になるまで繰り返す	LDIR	ED B0
CP s	Aレジスタの内容と8ビットデータsとの比較を行う	CP 10H	FE 10

命令のまとめ7

5.10 画面に模様を描く



マシン語で実行すればグラフィックもこの速さ

リスト16

```
                     ;
                     ;     SAMPLE
                     ;        PLOT SLANT LINES
                     ;
C000                 GVRAM: EQU   0C000H
00FE                 GENDH: EQU   0FEH
007F                 GENDL: EQU   7FH            G-VRAMの先頭アドレス
                     ;
005E                 GVRAM2:EQU   5EH            グラフィック画面2の選択ポート・アドレス
005F                 MRAM:  EQU   5FH            メインRAM選択ポート・アドレス
                     ;
                            ORG   0BC00H
                     ;
BC00 F3                     DI                   割込み禁止
BC01 3E80                   LD    A,80H          Aを“10000000”とする
```

```
BC03 01FFFF             LD    BC,0FFFFH    ループをまわる数をセットする
BC06 115000             LD    DE,80        VRAMのX方向のバイト数
BC09 2100C0             LD    HL,GVRAM     HLにG-VRAMの先頭アドレスをセットする
BC0C D35E               OUT   (GVRAM2),A   グラフィック画面2(緑)を選択する
                ;
BC0E F5         MAIN:   PUSH  AF           Aを退避する
BC0F AE                 XOR   (HL)         AとHLで示される画面の内容とのXORをとる
BC10 77                 LD    (HL),A       HLで示される画面にAを出力する
BC11 19                 ADD   HL,DE        次のアドレスを算出する
                ;
BC12 7C                 LD    A,H          }Hと G-VRAMの最終アドレスの上位8ビットとを比較する
BC13 FEFE               CP    GENDH
BC15 2805               JR    Z,SKIP       もし算出したアドレスの上位(H)がG-VRAMの最終アドレスの上位だったらSKIPへ飛ぶ
BC17 F23ABC             JP    P,UP         もし算出したアドレスの上位(H)がFFだったらUPへ飛ぶ
BC1A 1808               JR    NEXT         NEXTへ飛ぶ
BC1C 7D         SKIP:   LD    A,L          }LとG-VRAM 最終アドレスの下位8ビットとを比較する
BC1D FE80               CP    GENDL+1
BC1F F23ABC             JP    P,UP         もし算出したアドレスの下位(L)が7Fより大きかったらUPへ飛ぶ
BC22 1805               JR    CONT         CONTへ飛ぶ
                ;
BC24 FEBF       NEXT:   CP    0BFH         }もし算出したアドレスの上位(H)が(G-VRAM の先頭アドレ
BC26 CC40BC             CALL  Z,DOWN       }スの上位)-1 だったらDOWNへ飛ぶ
                ;
BC29 0B         CONT:   DEC   BC           BC=BC－1 (各フラグには無影響)
BC2A 78                 LD    A,B
BC2B B1                 OR    C            }BC=0だったらESCAPEへ飛ぶ
BC2C 2807               JR    Z,ESCAPE
                ;
BC2E F1                 POP   AF           Aの退避を解除する
BC2F 0F                 RRCA               Aの各ビットを1ビット分右へずらす(ビット・パターンを作る)
BC30 30DC               JR    NC,MAIN      キャリーでなかったら MAIN へ飛ぶ
BC32 23                 INC   HL           HL=HL+1 (アドレスの算出)
BC33 18D9               JR    MAIN         MAIN へ飛ぶ
                ;
BC35 F1         ESCAPE:POP    AF           スタック・ポインタの値を戻すためのダミー
BC36 D35F               OUT   (MRAM),A     メインRAMを選択する
BC38 FB                 EI                 割込み許可
BC39 FF                 RST   38H          モニタへ戻る
                ;
BC3A 11B0FF     UP:     LD    DE,-80       DE=－80
BC3D 19                 ADD   HL,DE        アドレスを補正する
BC3E 18E9               JR    CONT         CONTへ飛ぶ
                ;
BC40 115000     DOWN:   LD    DE,80        DE=80
BC43 19                 ADD   HL,DE        アドレスの補正
BC44 C9                 RET                リターンする
                ;
BC45                    END
```

このプログラムでは，図5.15に示す順にG-VRAM上の座標を移動しながら点を打っていきます．このプログラムで重要なのは，G-VRAM上でのアドレス計算です．図5.15に示すA点からB点にかけてのアドレスの計算は，HLレジスタ・ペアにC000H番地を，DEレジスタ・ペアに80をセットして，ADD DE, HLを実行することによって現在の座標のアドレスの下の座標アドレスを計算し，8回に1回INC HLを行うことによって点を打ちます(図5.16参照)．

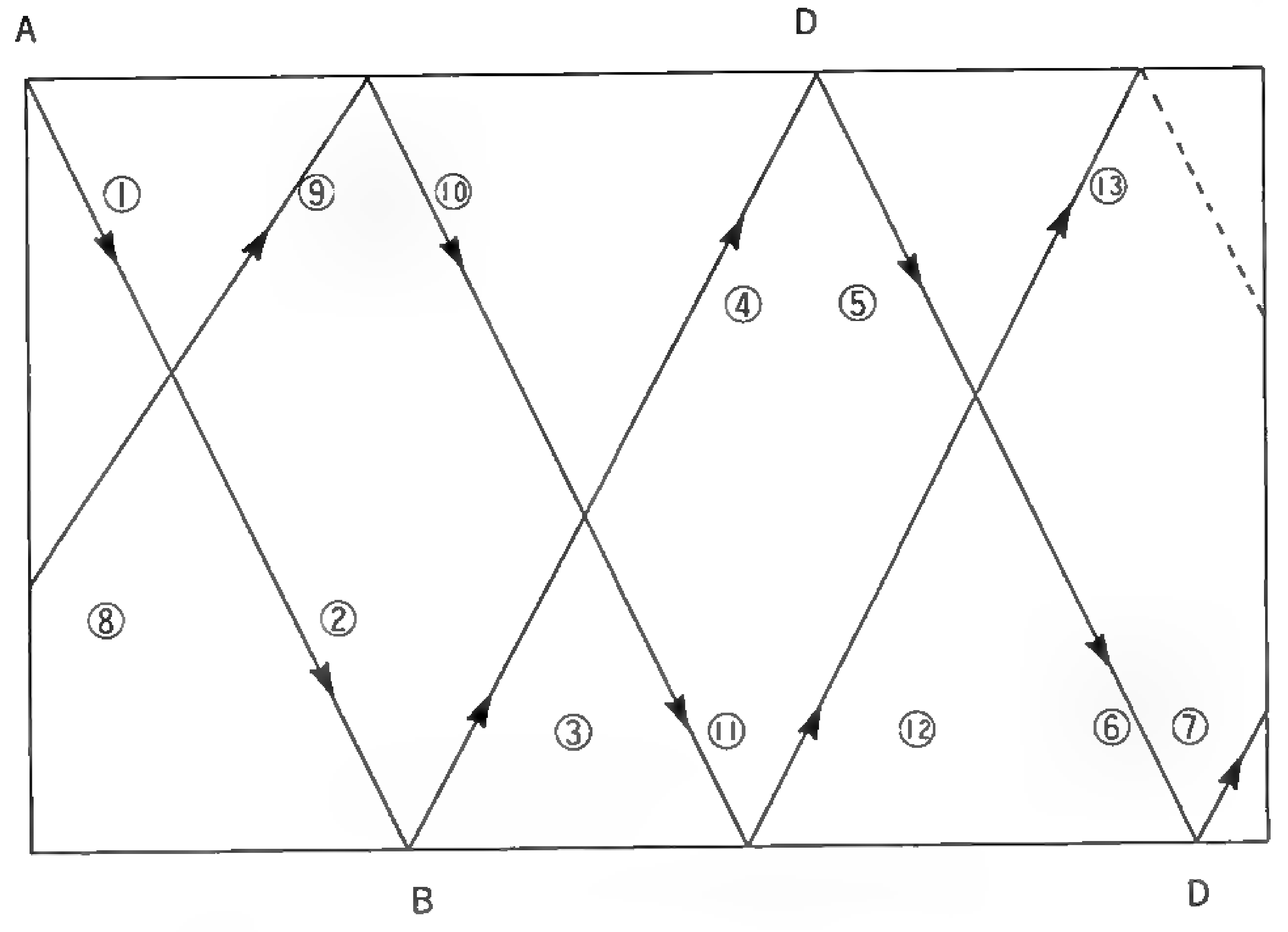

A
D
①
⑨
⑩
⑬
④
⑤
⑧
②
③
⑪
⑫
⑥
⑦
B
D

図5.15　画面模様を描く順序

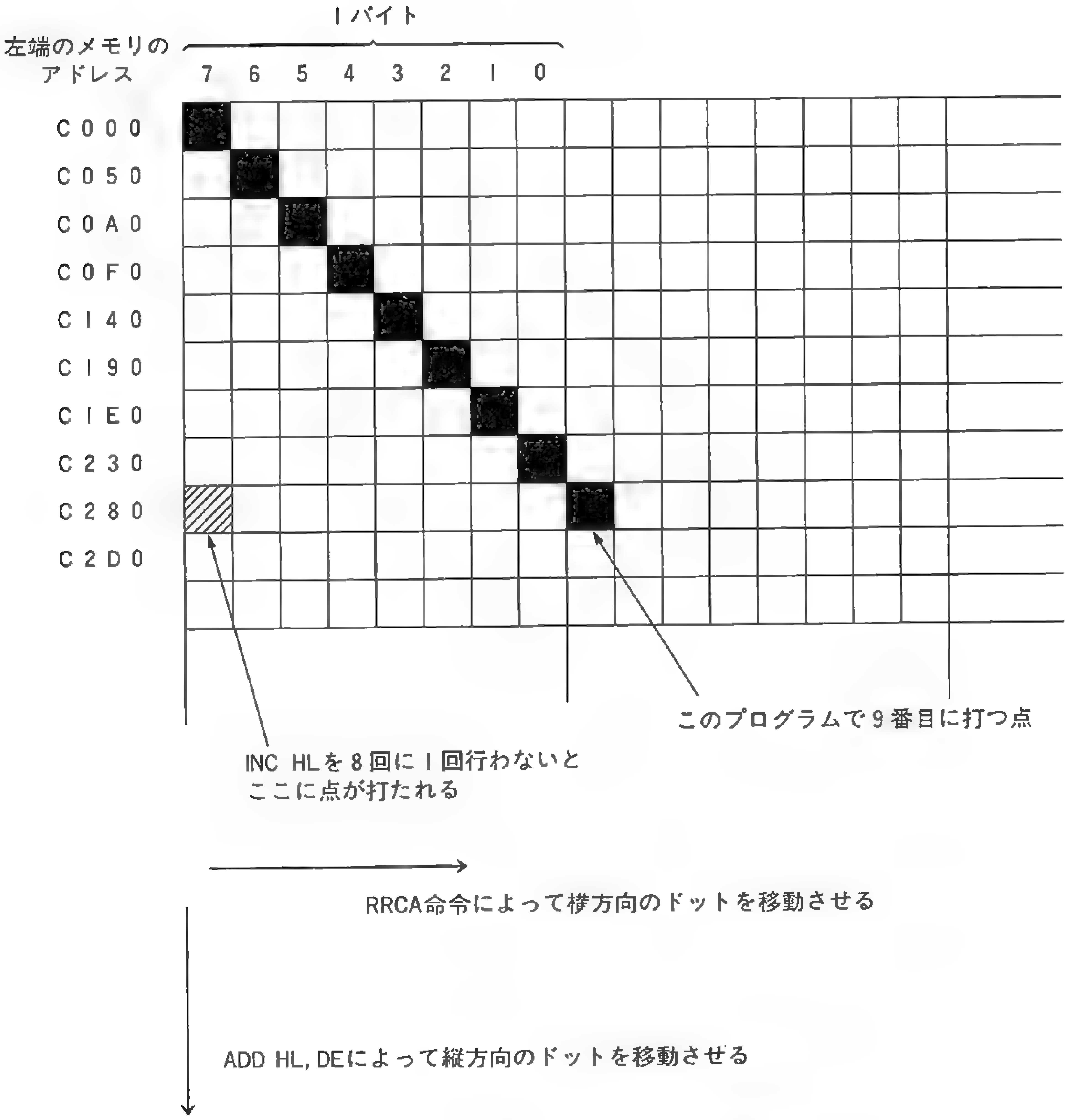

１バイト
左端のメモリの
アドレス
7
6
5
4
3
2
1
0
C000
C050
C0A0
C0F0
C140
C190
C1E0
C230
C280
C2D0
このプログラムで９番目に打つ点
INC HLを８回に１回行わないと
ここに点が打たれる
RRCA命令によって横方向のドットを移動させる
ADD HL, DEによって縦方向のドットを移動させる

図5.16　グラフィック画面上のメモリ構成

さて，B点の折り返しの判定ですが，ADD HL,DEの結果HLレジスタ・ペアの値がG-VRAMの最終アドレス(FE7FH番地)より大きくなったかどうかで行います(BC12H番地～BC23H番地)．判断の結果G-VRAMの最終アドレスより大きかった場合，DEレジスタ・ペアに－80をセットしてADD HL,DEを実行することによって画面内のアドレスに納めます．－80は２の補数を利用して表現しています(BC3AH番地～BC43F番地)．同様に，B点からC点にかけてはDEレジスタ・ペアに－80がセットされていますので，同じプログラムを使用することによって上の座標(現在の点の座標を(X，Y)とすれば(X+1，Y－1))を計算します．そしてC点の折り返しの判断を行っているのがBC24H番地の部分です．

このプログラムは，１回点を打つごとにBCレジスタ・ペアの値をデクリメントし，この値が０になったときプログラムを終了します(BC29H番地～BC2DH番地)．

5.10.1 RRCA命令

リスト16では，初めAレジスタに80Hつまり２進数で10000000をセットし，この命令によってセットされているビットを１ビットずつずらして，XOR (HL)によってメモリ上に点を打ちます．８回に１回の割合でCYフラグがセットされますから，そのときにHLレジスタ・ペアの値をインクリメントするようにしています．

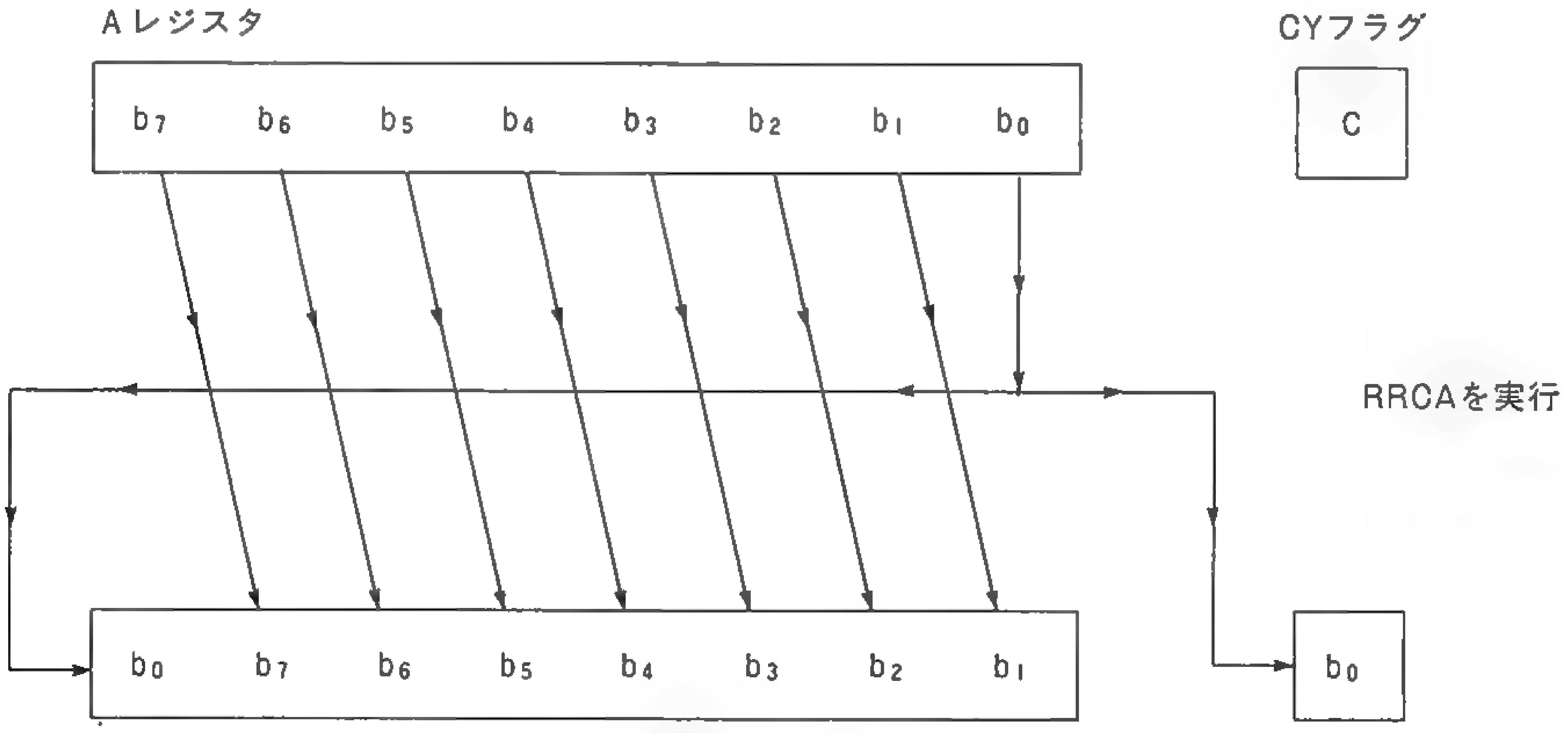


図5.17 RRCA

5.10.2 PUSH命令とPOP命令

PUSH／POP命令は，レジスタの内容を一時的に保存するために用いられます．ここではHLレジスタ・ペアを例に，これらの命令の動作を説明します．例えばプログラムの中で，HLレジスタ・ペアを利用したいがすでに他の用途に使用しており，現在のHLの内容も壊したくない場合にはPUSH HLを行います．

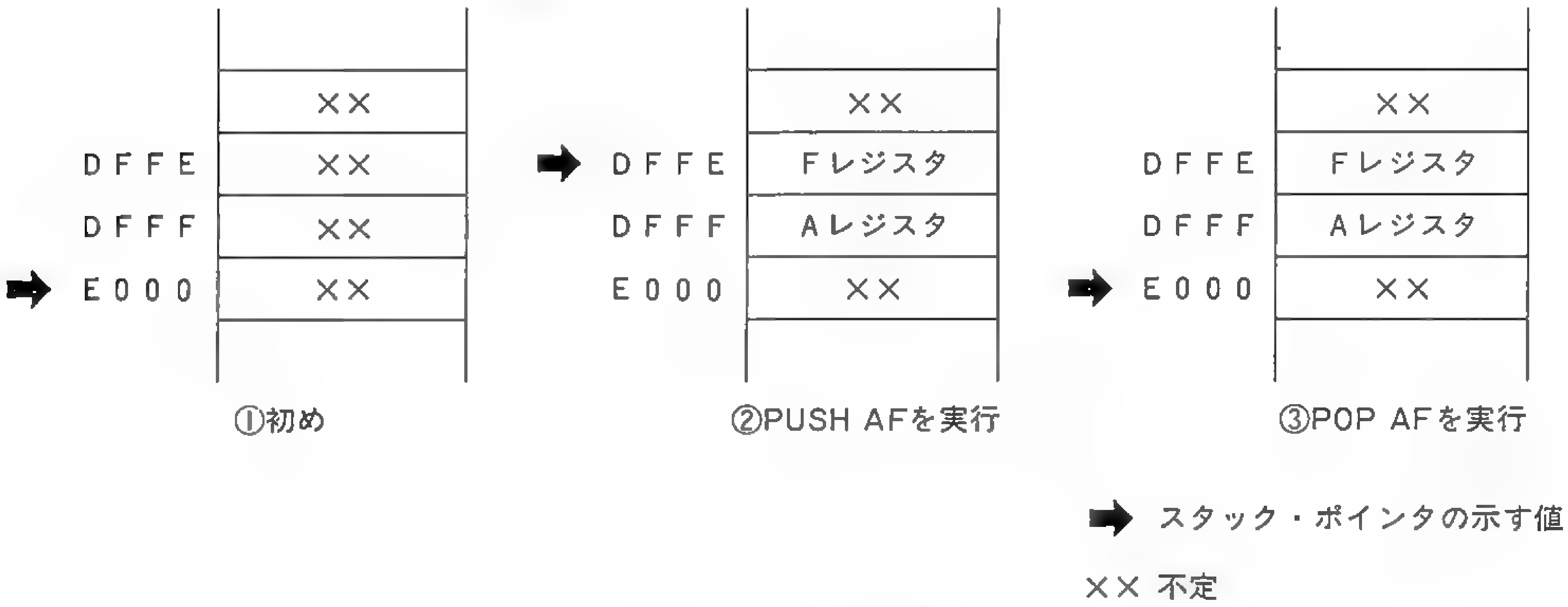
この命令をフェッチすると，CPUはスタック・ポインタの値をデクリメントしてスタック・ポインタの示すアドレスのメモリにHレジスタの値をロードし，次にスタック・ポインタを再びデク

リメントしてその値のアドレスのメモリにLレジスタの値をロードします．その結果，スタック・ポインタの値は命令の実行前に較べて－2され，HLレジスタ・ペアの内容はスタックに退避されたことになります．

さて，退避したデータを再度復帰する命令がPOPです．先程PUSHしたデータを再びHLレジスタ・ペアに戻す場合にはPOP　HLを行います．この命令をフェッチすると，CPUはスタック・ポインタの示すアドレスのメモリの値をLレジスタにロードしてスタック・ポインタをインクリメントします．次に，スタック・ポインタの示すアドレスのメモリの値をHレジスタにロードしてスタック・ポインタの値をインクリメントします．スタック・ポインタの値は，命令の実行前に較べて＋2されます．

リスト16では，PUSH AFとPOP AFを使用していますが，これはAレジスタの内容（１ビットだけ１で他の７ビットは０）が，他の作業によって破壊されるのを防いでいます．

スタック・ポインタが初めＥ０００の場合



①初め　②PUSH AFを実行　③POP AFを実行

図5.18　スタック領域の変化

5．10．3　点を打つ

リスト16では，XOR　(HL)命令によってHLレジスタ・ペアの示すメモリ，つまりグラフィック画面上に点を打ちます．

ここで，排他的論理和の性質を考えていただけばおわかりいただけると思いますが，この操作を２回行うと必ずもとのデータが得られます．つまり，このプログラムを２回実行すると，画面はもとに戻るのです．もとのデータをa，aと排他的論理和を２回とるデータをb，結果をyとして式で証明してみましょう．bは１でも０でもかまいません．

$$
\begin{aligned}
y &= (a \forall b) \forall b \\
&= a \forall (b \forall b) \quad (\because \text{排他的論理和において結合法則が成り立つ}) \\
&= a \forall 0 \\
&= a
\end{aligned}
$$

この式により，yはaに等しくなります．具体的な例を図5.19に示します．

	1 0 1 0 1 0 1 0	もとのデータ
XOR)	1 1 0 0 1 1 0 0	XORをとるデータ
XORを1回実行	0 1 1 0 0 1 1 0	
XOR)	1 1 0 0 1 1 0 0	XORをとるデータ
XORを2回実行	1 0 1 0 1 0 1 0	もとのデータ

図5.19 XORを2回とると

ニーモニック	命令の動作	命令のサンプル	
		ニーモニック	マシン語
RRCA	Aレジスタの内容のビット7をビット6に，ビット6をビット5というように各ビットをずらしてビット0をCYフラグとビット7に入れる	RRCA	0F
PUSH qq	qqレジスタ・ペアの内容をスタックに退避する	PUSH BC	C5
POP qq	PUSH命令によってスタック領域に退避したレジスタの内容をqqレジスタ・ペアに復帰する	POP BC	C1
XOR (HL)	Aレジスタの内容とHLレジスタ・ペアの示すメモリの内容との排他的論理和をとり，結果をAレジスタに入れる	XOR (HL)	AE
JR NC,e	CYフラグがリセットされているとき，PCの内容にe-2を足した値で示すアドレスへジャンプする	JR NC,e	30 e-2
JP P,nn	Sフラグがリセットされているときnnで示されるアドレスへジャンプする	JP P,FF00H	F2 00FF
CALL Z,nn	Zフラグがセットされているときnnをスタート・アドレスとするサブルーチンを呼び出す	CALL Z,FF00H	CC 00FF

命令のまとめ8

5.10.4 JP命令

今まで説明してきた分岐命令はすべてプログラム・カウンタにディスプレイスメント－2を加えた値のアドレスに分岐する命令であったのに対し，この命令はオペランドとして直接分岐先をもちます．

JR eに代表される，オペランドとしてディスプレイスメントをもつ分岐命令を相対分岐命令と呼びましたが，JP命令のように，オペランドとして分岐先のアドレスをもつ分岐命令を絶対分岐命令と呼びます．相対分岐命令がプログラム・カウンタの示す値を中心に＋127から－128の範囲しか分岐先が選べないのに対し，絶対分岐命令は全アドレス，つまり0000H番地からFFFFH番地の任意のアドレスに分岐できるのが特徴です．反面，あるプログラムの分岐命令がすべて相対分岐命令だけで書かれていれば，プログラムのスタート・アドレスを変更してもマシン語プログラムの書き換えの必要はありません．しかし絶対分岐命令を使用した場合はマシン語プログラムの分岐先を変更しなければなりません．前者のようなプログラムのスタート・アドレスを自由に選べるプログラムをリロケータブル(relocatable)なプログラムといいます．

5.11　ハンドアセンブルの方法

今までMF-ASMを利用してアセンブルしてきたにもかかわらず，ここでハンドアセンブルの方法を説明するのはおかしなことだと思われるかもしれません．しかし本来ハンドアセンブルで行っていたことをプログラムに置き換えたものがアセンブラですから，ハンドアセンブルの方法を知ることはアセンブラの働きを理解する参考になると思いますので一通り説明することにします．

ハンドアセンブルは，プログラムサイズ(プログラムの大きさ)が小さい場合にはたいしたことはないのですが，少しプログラムが大きくなるとたいへんな労力と時間が必要になる仕事です．

ハンドアセンブルによるマシン語プログラムの作成は，一般に次のような過程で行います．

(1)　ソース・プログラムを作成する
(2)　デバッグ(虫とり)を充分に行う
(3)　ニーモニック↔マシン語対照表(インストラクション表)をみてニーモニックをマシン語に変換する(ハンドアセンブル)

(1), (2)の過程は，アセンブラを使用してプログラム開発を行う場合にも同じことが必要であり，(3)の過程のマシン語変換のみが，自分でやるかプログラム(アセンブラ)に頼るかの違いがあるのです．

では上記の3つの過程を個別に説明しましょう．

(1)　これをやらないことにはプログラムを作ることはできません．ニーモニックの学習に不充分な点があると，実際にない命令を作ってしまいます．
(2)　この過程は非常に重要です．ハンドアセンブルする場合は，一度マシン語に変換してしまうとプログラムに誤りがあったときの修正が困難になります．ですから，(3)のハンドアセンブルに移る前によくチェックしてください．
(3)　マシン語に変換するときは，インストラクション表をよくみて，間違いのないよう注意します．いくらソース・プログラムが正しくても，ここで間違えていたのではどうにもなりません．最後に，でき上ったプログラムをモニタを使って入力して実行すればよいのです．

さて，それでは実際のプログラミングに入りましょう．

ハンドアセンブルを行う場合は，プログラムの書式には，いっさい制限はありません．ただ自分がわかりやすいように書けばよいのです．書式に制限を受けるのはアセンブラを使用するときです．ただ，両者の間であまりに書式が違うとわかりづらくなりますから，一定の書式で書くことをお勧めします．

表5.2をみてください．これは，1から100までを加えてその結果をDEレジスタ・ペアにしまうプログラムのソース・リストです．私の場合は，この表のような枠線の紙を使用しました．この

ように，プログラムを書くための紙をプログラム・コーディングシート(Program Coding Seet)といいます．

アドレス	マシン語	ラベル	ニーモニック	コメント
		START	EQU　0C000H	
			ORG　START	プログラムのスタートアドレスは0C000H
			XOR　A	Aレジスタをクリアする
			LD　D, A	Dレジスタをクリアする
			LD　C, 1	Cレジスタに1をロード
			LD　B, 100	Bレジスタに100をロード
		LOOP	ADD　A, C	AレジスタとCレジスタを加える
			JP　NC, SKIP	CYフラグが0ならSKIPへ飛ぶ
			INC　D	Dレジスタに1を加える
		SKIP	INC　C	Cレジスタに1を加える
			DJNZ　LOOP	100まで足したか否かを判定する
			LD　E, A	EレジスタにAレジスタの内容をロードしDEレジスタが答えとなる
			RST　38H	モニタに戻る
			END	

表5.2　ソース・プログラムの作成

これは，アドレス，マシン語，ラベル，ニーモニック，そしてコメントの5つの欄から成り立ちます．先程の(1)および(2)の段階で使用するのは後の3つの欄です．ラベルの欄は，分岐(Jump)命令のための分岐先や数値に名前付けをするための欄です．次のニーモニックの欄は，ニーモニックを書くための欄，そしてコメントの欄は注釈を書くための欄です．マシン語のプログラムを書く際に，このコメントは非常に重要です．BASICなどとは違って，コメントがないと忘れたころに見直した場合にこのプログラムは何をやっていたのかわからなくなってしまうことが多いので，コメントがあれば，後からプログラムを見直したり，デバッグする際に随分と手助けになり，他人にプログラムをみせるときに，その人の理解を助けることにもなります．

表5.2のように，ソース・プログラムが完成したら，(3)の過程に入ります．

これには付録のインストラクション表を利用します．例としてLD命令についてみてみましょう．

表5.3に，8ビットのロード命令のインストラクション表の一部を示します．さて先程のプログラムの4，5，6，12行目にロード命令が使ってありました．**表5.3**からLD D, Aのマシン語は57H, LD E, Aのマシン語は5FHとなります．さて，5行と6行の命令は，どちらも2バイト命令で，1バイト目がオペ・コード，LD B, 100なら，オペ・コードは，06Hとなります．また，**表5.3**の同じ枠の中にnとありますが，nは8ビット(1バイト)のデータを表します．LD B, 100の場

合のnにあたるのは100というわけです．ただし，マシン語の場合は10進数では表現できませんから，16進数に変換しなければなりません．100を16進数に変換すると64Hとなります．ですから，LD B,100をマシン語に変換すると，06 64Hと２バイトになるのです．同様にしてLD C,1は0E 01Hとなります．

×	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	(nn)	n	I	R
LD A,×	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n	ED 57	ED 5F
LD B,×	47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n		
LD C,×	4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n		
LD D,×	57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n		
LD E,×	5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n		
LD H,×	67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n		
LD L,×	6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n		
LD (HL),×	77	70	71	72	73	74	75							36 n		
LD (BC),×	02															
LD (DE),×	12															

表5.3　LD命令のインストラクション表

次のADD A,Cもインストラクション表をみれば一目瞭然でしょう．さて８行目の命令はJP NC,SKIPとあり，オペ・コードのD2Hというのはわかりますが，SKIPのアドレスが決まっていないので，ここはひとまず２バイト抜かしておいて後で埋めることにしますINC D／INC Cまでくれば，それぞれ14H，0CHと各々１バイトなのでSKIPはC00BH番地に決まり，JP NCの分岐先のアドレスが入りました．次の命令はDJNZ LOOPです．命令表をみましたか?$\genfrac{}{}{0pt}{}{10}{e-2}$となっています．オペ・コードの10Hはよいのですが，eは次の式で表されます．

e＝〈分岐先のアドレス〉－〈DJNZの10Hのアドレス〉

つまり，この例でいうと

e＝C006H－C00CH
　＝FAH(２の補数)

さらにオペランドの値は02Hを引いていますから

e －02H＝FAH－02H＝F8H

となります．普段アセンブラを利用していると，相対分岐先のアドレスなど気にしなくてよいのですが，ハンドアセンブルで記述する上では一番大事なことですからしっかり覚えてください．最後にハンドアセンブルの完成したものを載せておきます．

アドレス	マシン語	ラベル	ニーモニック	コメント
		START	EQU 0C000H	
			ORG START	プログラムのスタート・アドレスは0C000H
C000	AF		XOR A	Aレジスタをクリアする
C001	57		LD D, A	Dレジスタをクリアする
C002	0E 01		LD C, 1	Cレジスタに1をロードする
C004	06 64		LD B, 100	Bレジスタに100をロードする
C006	81	LOOP	ADD A, C	AレジスタとCレジスタを加える
C007	D2 0B CD		JP NC, SKIP	CYフラグが0ならばSKIPへ飛ぶ
C00A	14		INC D	Dレジスタに1を加える
C00B	0C	SKIP	INC C	Cレジスタに1を加える
C00C	10 F8		DJNZ LOOP	100まで足したか否かを判定する
C00E	5F		LD E, A	EレジスタにAレジスタの内容をロードしDEレジスタが答えとなる
C010	FF		RST 38H	モニタに戻る
			END	

表5.4　ハンドアセンブルの例

第6章　マシン語プログラム・テクニック

前章までで，μPD780のマシン語のほとんどの命令を紹介しました．サンプル・プログラムの中にはかなり複雑なものもありますから，この時点ですべて理解している必要はありません．ただ，個々の命令の働きについては，インストラクション表を参照すれば理解できるようにしておいてください．

プログラム・テクニックは，初期の段階では命令の使い方に重点がおかれますが，命令に慣れてくればプログラムのアルゴリズムという，BASICや他の高級言語，CPUにも共通するような点に集約されてきます．もちろんアルゴリズムの中で，特定の言語，CPUのアーキテクチャに見合ったテクニックを駆使することは必要であり，場合によってはメモリの制約やスピードの点から特別なテクニックを使わざるを得ないこともあります．

この章のプログラムは，今までに説明した命令を組み合わせて使用した基本的なものです．これらは実際にもよく使うプログラムですから，応用できるようにしておくと他のCPUを使うときに役に立ちます．

6.1　危険なマシン語!?

BASICのプログラムでいわゆる「暴走」ということを経験したことはありますか．プログラムを実行したらうまく動かず，画面には変な模様がでて，キー入力は受け付けられないという悲惨な状態です．BASICではある程度のエラー処理はインタプリタが行ってくれますが，マシン語ではメモリに書かれたプログラムをそのままCPUが実行していきますから，1バイトでも不適当なデータが書かれていれば暴走することがあります．また，図6.1のようにスタート・アドレスが違っただけでも暴走することがあります．

暴走の原因としては，次のようなことが考えられます．

- システム領域の書き換え
- スタック領域の破壊

●割込みに関係する部分の書き換え

その他，無限ループも一種の暴走とみなすことがあります．

暴走は，リセットすることによって一応解決するのですが，せっかく入力したプログラムを消してしまったり，ひどい場合には勝手にディスクが動き出して，ディスケットに書かれているデータを壊すことがありますから注意が必要です．

動作の確かでないマシン語プログラムは，実行前にセーブ(BSAVE, Wコマンドなど)しておくことはもちろん，ディスクのフタを開けておくくらいの用心をした方がよいでしょう．

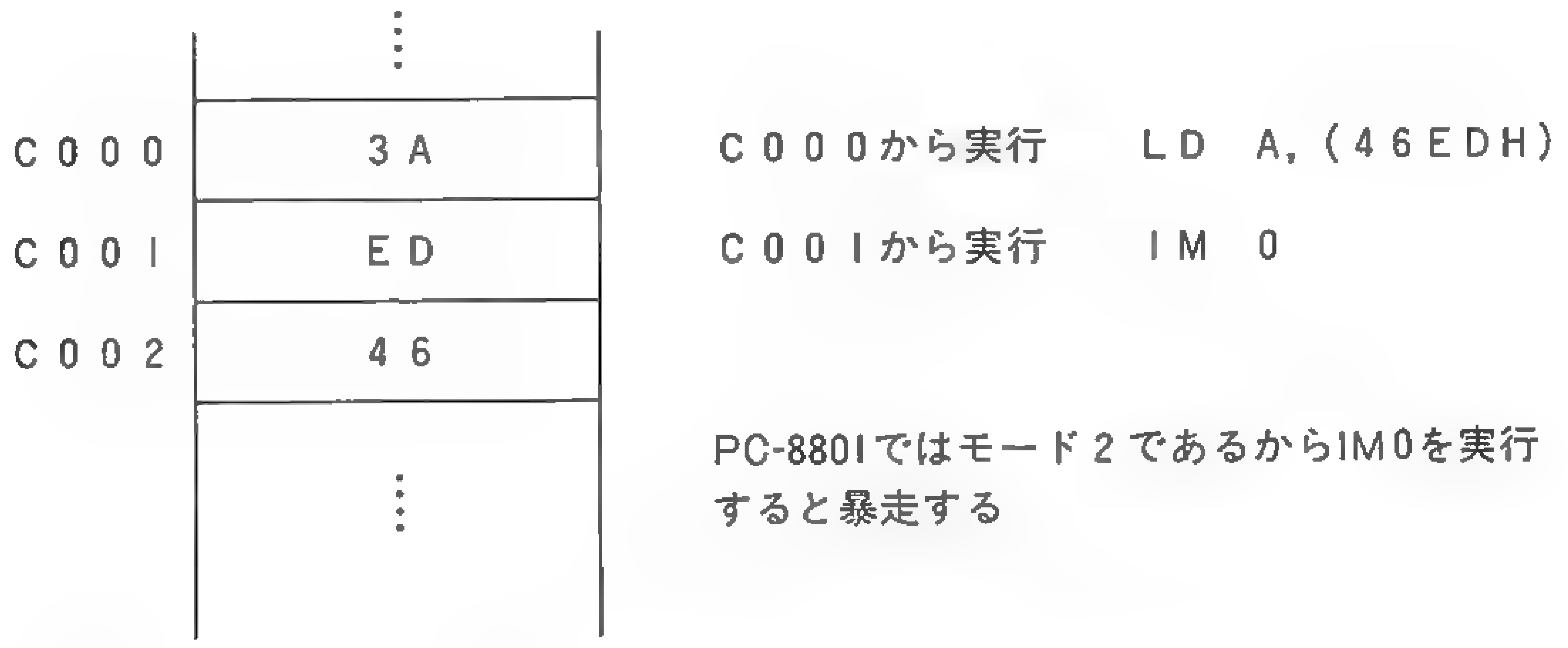

C000
C001
C002
3A
ED
46
C000から実行　LD A,(46EDH)
C001から実行　IM 0
PC-8801ではモード2であるからIM0を実行すると暴走する

図6.1　スタート・アドレスによって異なった命令になる例

［注］リセットの方法について

ハードウェア的にリセットを行う方法として次の3つがあります．

(1)　電源を一度切って入れ直す
(2)　リセットボタンのみを押す
(3)　STOP を押しながらリセットボタンを押す(ホットスタート)

(3)の方法はプログラム領域が保存されるため，マシン語プログラムが無限ループになっているときなどに有効です．

6.1.1　マシン語プログラムのデバッグ

BASICと違って，マシン語のプログラムを作成するときはアセンブラを使う場合もハンドアセンブルをするときと同じように，まず紙などにニーモニックで書いてから打ち込むようにします．これは，他の多くの言語についてもいえることですが，BASIC(インタプリタ)のように対話形式でプログラムを作っていくと，かえって時間がかかることがあるからです．

マシン語プログラムのデバッグ方法には，机上でリストを眺めて行うものと，実際に少しずつ実行しながら行う方法があります．後者は，ブレークポイント・アドレスを設定しながら実行していくもので，BASICでのSTOP命令を用いるデバッグの方法に似ています．

具体的には，モニタのGコマンドでブレークポイント・アドレスを設定して実行し，Xコマンドでレジスタの内容を調べたり，Dコマンドなどでメモリの内容を調べることによって動作を確認していきます．この方法では，ニーモニック・レベルではなくマシン語レベルで16進数を眺めながらデバッグしますから，ハンドアセンブルでこれらに慣れておくとデバッグに要する時間を短縮することができます．

6.2　BASICプログラムとのリンク

6.2.1　USR関数の使い方

BASICプログラムとマシン語プログラムをリンクする方法として，N_{88}-BASICには2種類のコマンドが用意されています．それがUSR関数とCALL文(CALL文はN_{88} DISK-BASICでのみ利用可)です．

それでは，USR関数から説明しましょう．

USR関数とは，使用者つまりユーザ(User)が定義できる関数という意味でこの名前があります．では，DEF FN文で定義した関数とどう違うのかといいますと，DEF FN文がBASICの関数や式を用いて定義するのに対して，USR関数はユーザが作成したマシン語プログラムを呼び出します．また，USR関数は，関数であるが故に主に代入文で使用されます．書式は次の通りです．

(書式)　USR [〈番号〉] (〈引数〉)

〈番号〉は，0～9の数値であり，省略した場合は0となります．これはUSR0～USR9までの10種類のUSR関数を1つのBASICのプログラム中で使用できることを表しています．

また，関数であることから必ず引数を必要とします．この引数の受け渡しは，文字型変数を除いて，FAC(浮動小数点アキュムレータ)を通じて行われます．FACは，BASICインタプリタが計算のために用いるメモリ上に設定された仮想のアキュムレータであり，USR関数の結果も，このFACを通じて引き渡されます．

ところでUSR関数を使用するに先立って，マシン語プログラムのスタート・アドレスを決めなければなりません．これはBASICが勝手に決めるわけではなく，ユーザがプログラム中で宣言します．そのための文がDEF USR文です．書式は次の通りです．

(書式)　DEF USR [〈番号〉]= 〈スタート・アドレス〉

DEF USR文はUSR関数1つずつに対して，必ず指定されます．例えば，アドレスE000H番地から始まるマシン語プログラムにおいて，引数を0としたときの結果を知りたい場合は，次のようになります(変数はAでなくてもよい)．

```
10  DEF USR=&HE000
20  A=USR(0)
30  PRINT A
40  END
```

もちろん，このプログラムの実行に先立ち，マシン語プログラムを作成，入力しておかなければなりません．

次に，USR関数およびCALL文で用いるマシン語のプログラム作成にあたっての注意を述べます．

(1)　ユーザのマシン語プログラムの終了はRET命令でなければならない
(2)　スタック・ポインタの値は，マシン語プログラムの終了時に，呼び出されたときと同じ値でなくてはならない
(3)　マシン語プログラム用として，スタックは16バイト利用できるようになっている

(3)について説明を加えるとマシン語プログラム中でPUSH／POPを使用しなければ，サブルーチンからサブルーチンを呼ぶネストは8回まで許されています．もし，それ以上のスタックを使用する場合はスタック・ポインタに新たな値を設定する必要がありますが，マシン語プログラムの最後に必ずスタック・ポインタの値をもとに戻さなければなりません．この操作を行わないと暴走することがあります．

以上がマシン語プログラム使用時の注意点です．

次に引数の受け渡しについて説明します．USR関数の場合，引数はFACを通じて受け渡されると述べました．しかし，FACは一体どこにあるのでしょう．これについては何ら気にする必要はなく，ユーザのマシン語プログラムが呼び出されたとき，HLレジスタ・ペアがFACのある場所を示していますので，HLレジスタ・ペアを基準にプログラムの中で引数を受け取ればよいのです．FACは，倍精度演算を行うため8バイトあります．そのうち，単精度演算を行う場合は4バイト，整数演算を行う場合は2バイトを使用します．そのため，USR関数で引数を受け渡した場合，その引数の型によってFACに入る形態が変わり，ユーザのマシン語プログラムが呼び出されたときAレジスタには数値の型を示す数が入っています．これによって，どのような型の数がFACに入っているかを知ることができます．つまりマシン語プログラムでは，HLレジスタ・ペアとAレジスタの内容を調べることによって引数を知ることができます(図6.2，図6.3参照)．

今までは数値変数での話でしたが，引数に文字変数を使用した場合はどうなるのでしょう．文字変数は，数値変数のように使用するメモリ量が一定ではありませんので，FACによって引き渡しを行うわけにはいきません．文字変数を引数とした場合は，ユーザのマシン語プログラムに入ったとき，Aレジスタの中には文字変数であることを示す数である3が入ります．また，DEレジスタ・ペア(HLではありません)がストリング・ディスクリプタ(文字データの格納エリアをさすポ

インタ・エリア)のあるアドレスを指します．この値は，VARPTR関数を使用して得られる値と同じです．この構成を，図6．4に示します．

変数の型	Aレジスタの値
文字列型	3
整数型	2
単精度実数型	4
倍精度実数型	8

図6.2　Aレジスタとデータ型

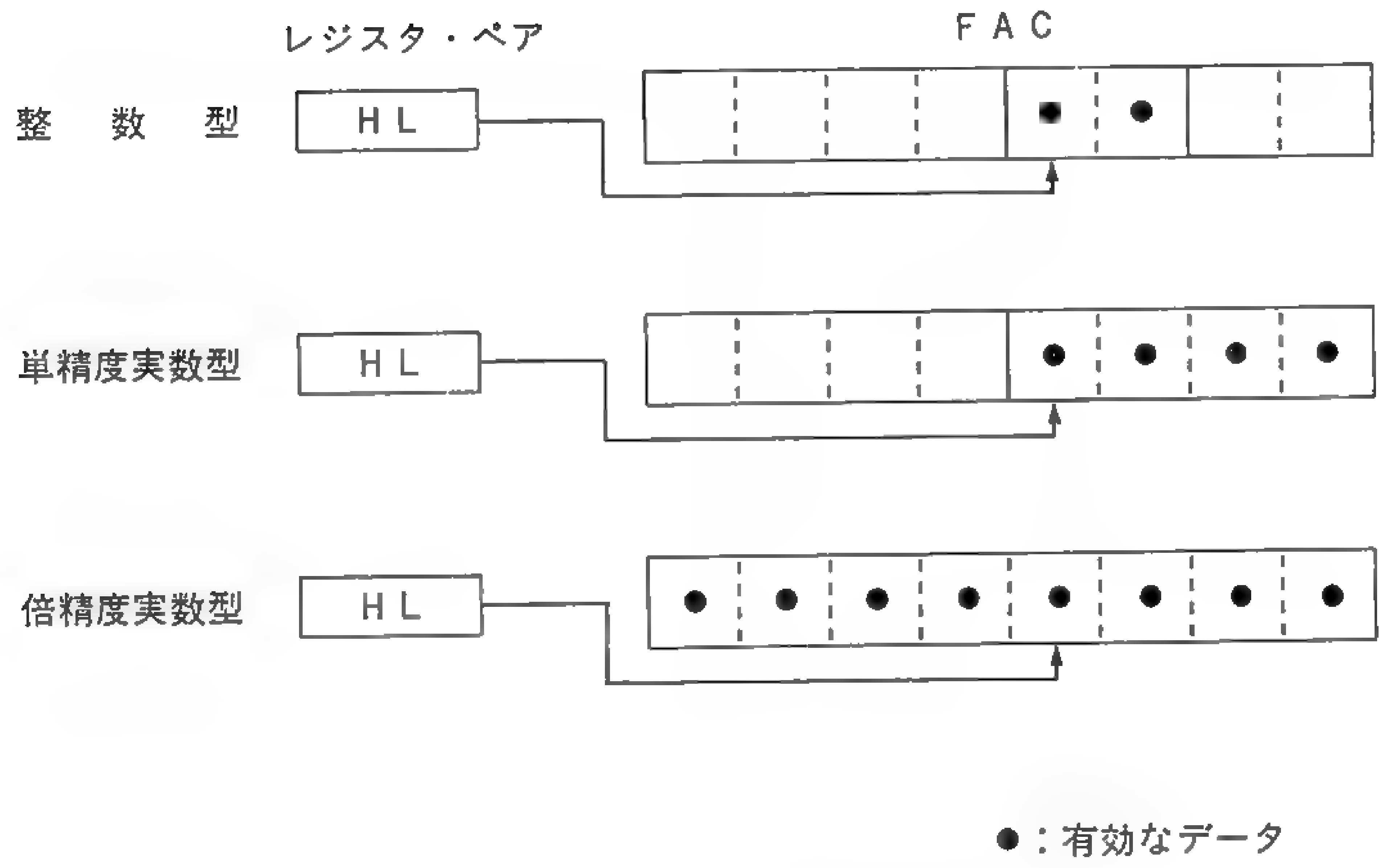

レジスタ・ペア
FAC
整数型
単精度実数型
倍精度実数型
HL
HL
HL
●：有効なデータ

図6.3　FACの収納状態

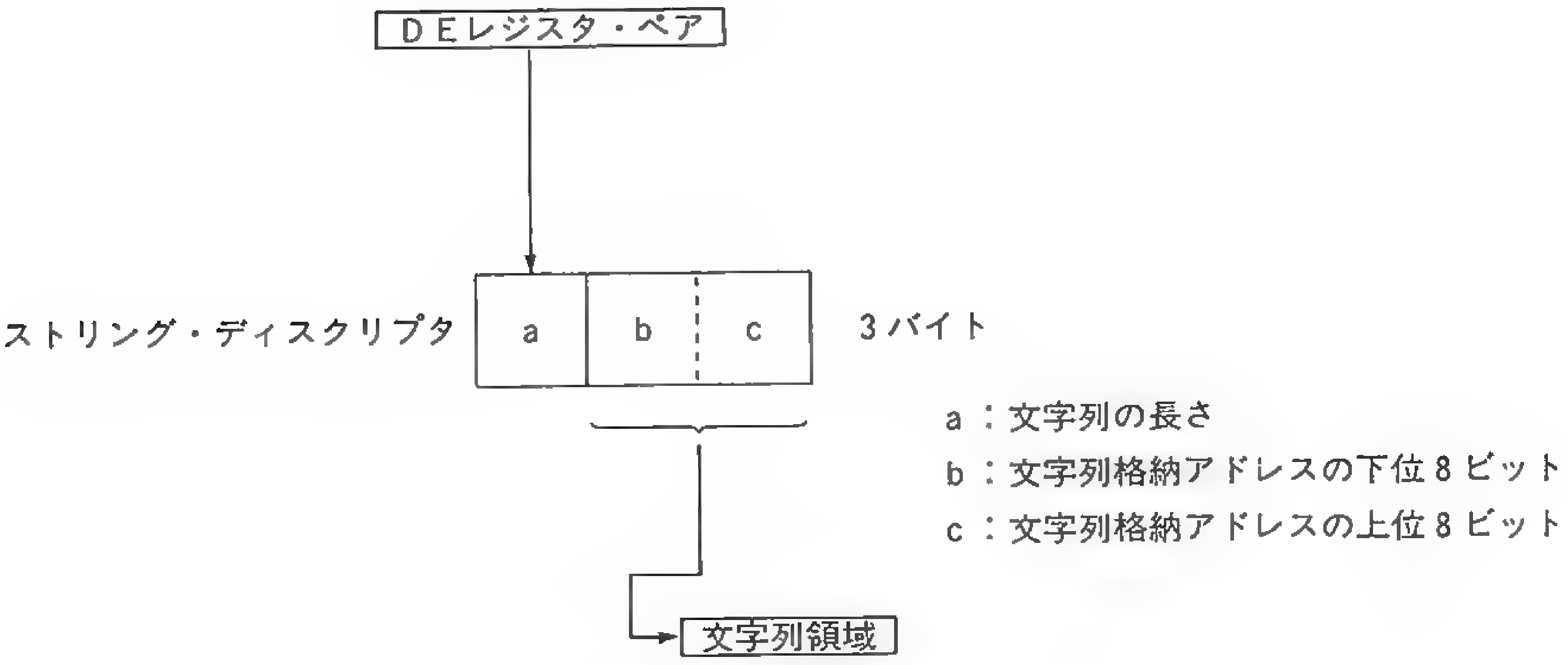

DEレジスタ・ペア
ストリング・ディスクリプタ
a
b
c
3バイト
a：文字列の長さ
b：文字列格納アドレスの下位8ビット
c：文字列格納アドレスの上位8ビット
文字列領域

図6.4　引数が文字列型であった場合

さて，マシン語のプログラムにおいては，引数を受け取ることができれば，数値変数の場合には必要に応じて結果をFACに戻せばよいのです．文字変数の場合は，文字変数自身を書き直します．ただし，この場合，ストリング・ディスクリプタの内容を書き換えたり文字列の長さを変えてはいけません．

このようなことからおわかりになると思いますが，USR関数に引数として代用した変数の型と，USR関数の型は一致するのです．ただし，１つだけ例外があります．それはBASICの内部ルーチンを利用することにより，値を整数とすることができることです．つまり，BASICに戻る前に，21FDH番地を呼ぶことにより，このときHLレジスタ・ペアに入っている値をUSR関数の値とします．

ところで，計算プログラムではなく単純な処理，例えば自分で作った画面クリア・プログラムなどだけを行いたいマシン語プログラムをUSR関数で呼び出すにはどうすればよいのでしょう．これは変数の受け渡しに関することに全く手を出さなければよいのです．また，変数をいくつも引き渡したい場合は，POKE文を使って直接メモリ上に書き込めばよいのです．以下にUSR関数を使用したプログラム例を示します．

リスト1

```
100 '
110 '   test usr function
120 '
130 CLEAR,&HDFFF : DEFINT I-N
140 GOSUB *INIT.MACHINE
150 DEF USR=&HE000
160 INPUT "string ";A$
170 DUMY$=USR(A$)
180 PRINT DUMY$,A$
190 END
200 '
210 *INIT.MACHINE
220 FOR IADR=&HE000 TO &HE020
230   READ A$ : POKE IADR,VAL("&H"+A$)
240 NEXT : RETURN
250 DATA FE,03,C0,EB,4E,79,A7,C8,3D,C8,23,5E,23,56,6B,62
260 DATA 06,00,09,2B,CB,39,1A,46,77,78,12,2B,13,0D,20,F6
270 DATA C9
```

```
                    ;
                    ;   TEST USR FUNCTION
                    ;
                            ORG   0E000H
                    ;
E000 FE03                   CP    3         渡された変数が文字列変数か否かチェックする
E002 C0                     RET   NZ        違っていれば何もせずにBASICへ戻る
E003 EB                     EX    DE,HL     DEレジスタの値をHLレジスタ・ペアへ移す
E004 4E                     LD    C,(HL)    }ストリング・ディスクリプタ中文字列の長さをCレジスタとAレジ
E005 79                     LD    A,C       }スタへロードする
```

```
E006 A7               AND  A        }
E007 C8               RET  Z        }文字列の長さが1または0ならば交換の必要がないのでBASIC
E008 3D               DEC  A        }へ戻る
E009 C8               RET  Z        }
                ;
E00A 23               INC  HL       }
E00B 5E               LD   E,(HL)   }DEレジスタ・ペアへ文字列領域の先頭アドレスをセットする
E00C 23               INC  HL       }
E00D 56               LD   D,(HL)   }
                ;
E00E 6B               LD   L,E      }
E00F 62               LD   H,D      }
E010 0600             LD   B,0      }HLレジスタ・ペアへ文字列領域の最終アドレスをセットする
E012 09               ADD  HL,BC    }
E013 2B               DEC  HL       }
E014 CB39             SRL  C        ループ回数を文字列の長さの1/2にセットする
                ;
E016 1A         LOOP: LD   A,(DE)   }
E017 46               LD   B,(HL)   }
E018 77               LD   (HL),A   }HL，DE各レジスタ・ペアの指し示すメモリの内容を交換する
E019 78               LD   A,B      }
E01A 12               LD   (DE),A   }
E01B 2B               DEC  HL       }HL，DEレジスタ・ペアを次に交換するメモリのアドレスにセッ
E01C 13               INC  DE       }トする
E01D 0D               DEC  C        }交換が終了したか否かを判定する
E01E 20F6             JR   NZ,LOOP  }
                ;
E020 C9               RET           BASICへ戻る
                ;
E021                  END
```

6．2．2　CALL文の使い方

BASICからユーザのマシン語プログラムを呼び出すもう１つの方法，CALL文について説明します．CALL文はN_{88} DISK-BASICのみでサポートされていますので，他のモードでは使用できないことをあらかじめ断わっておきます．

さて，CALL文ですが，USR関数とどこが違うかといいますと，それは複数の引数を受け渡しできることです．CALL文の書式は次の通りです．

CALL　〈変数名〉〔(〈引数〉［,〈引数〉…］)〕

最初の変数の値がマシン語プログラムのスタート・アドレスを示します．これは必ず数値変数であって，定数であったり，文字変数であってはいけません．また，配列変数であってもいけません．

引数は，CALL文の場合はいくつでも指定できます．ただし，やはりこれも変数でなければなりません．CALL文の書式は次のようなものがあります．

(1)　START＝&HE000：CALL START

(2)　START＝&HE000：A＝0：B＝100：CALL START (A, B)

(1)の場合は，E000H番地から始まるマシン語ルーチンを呼び出します．(2)の場合は，スタート・

アドレスは同じですが，引数として変数名AとB(この場合Aは0，Bは100)を引き渡します．

引数を使用しない場合(例(1))には全く支障はありませんが引数を渡す場合は，引数の個数によってそれぞれ状態が違います．

まず，引数が1個の場合はHLレジスタ・ペアが変数のある場所を示します．この場合のHLレジスタ・ペアの値は，VARPTR関数でその変数の値が格納されている場所を調べた値と等しくなります．引数が2つの場合は，1つ目の変数格納場所をHLレジスタ・ペアが，2つ目の変数格納場所をDEレジスタ・ペアが指しています．引数が3個までの場合は同様にして，3つ目の変数の格納場所をBCレジスタ・ペアが指しますが，引数が4個以上となると少し話が違ってきます．2つ目の変数までは，HLレジスタ・ペア，DEレジスタ・ペアがそれぞれ格納場所を直接示しますが，3つ目以降の変数に対しては，変数格納テーブルというものが作成され，BCレジスタ・ペアがそのテーブルの先頭アドレスを示します．これらのことを図6．5に示します．また，実際にプログラムを組んだものをリスト2に示します．

このプログラムでは，変数の受け渡しの参考例としてマシン語の部分は単純にメモリ中にある整数変数の値をワークエリアに移し，その値をBASIC部分で表示させているだけです．

その他にCALL文で注意しなければならないことは，変数の型がチェックされないので，変数型に関しては，必ず一致させる必要があることです．ついうっかり，整数型のつもりで計算を行っていたら，実は実数型であったなどということがありますから注意してください．

変数が3個まで

各レジスタペアの示す値

HL → 1番目の変数の格納アドレス
DE → 2番目の変数の格納アドレス
BC → 3番目の変数の格納アドレス

変数が4個以上

HL → 1番目の変数の格納アドレス
DE → 2番目の変数の格納アドレス
BC → 変数格納テーブル

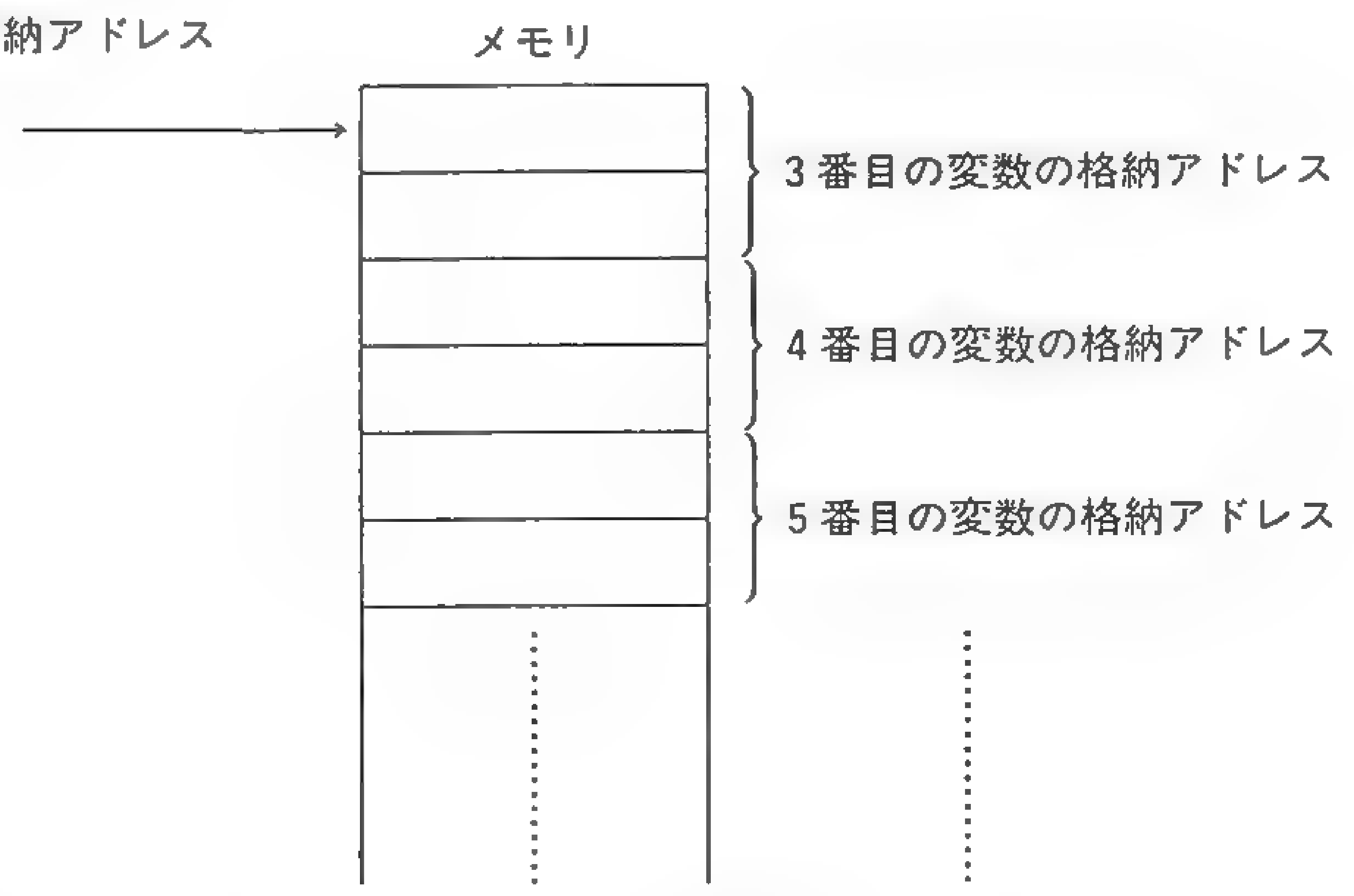

メモリ
3番目の変数の格納アドレス
4番目の変数の格納アドレス
5番目の変数の格納アドレス

図6.5 CALL命令とBCレジスタと変数格納テーブル

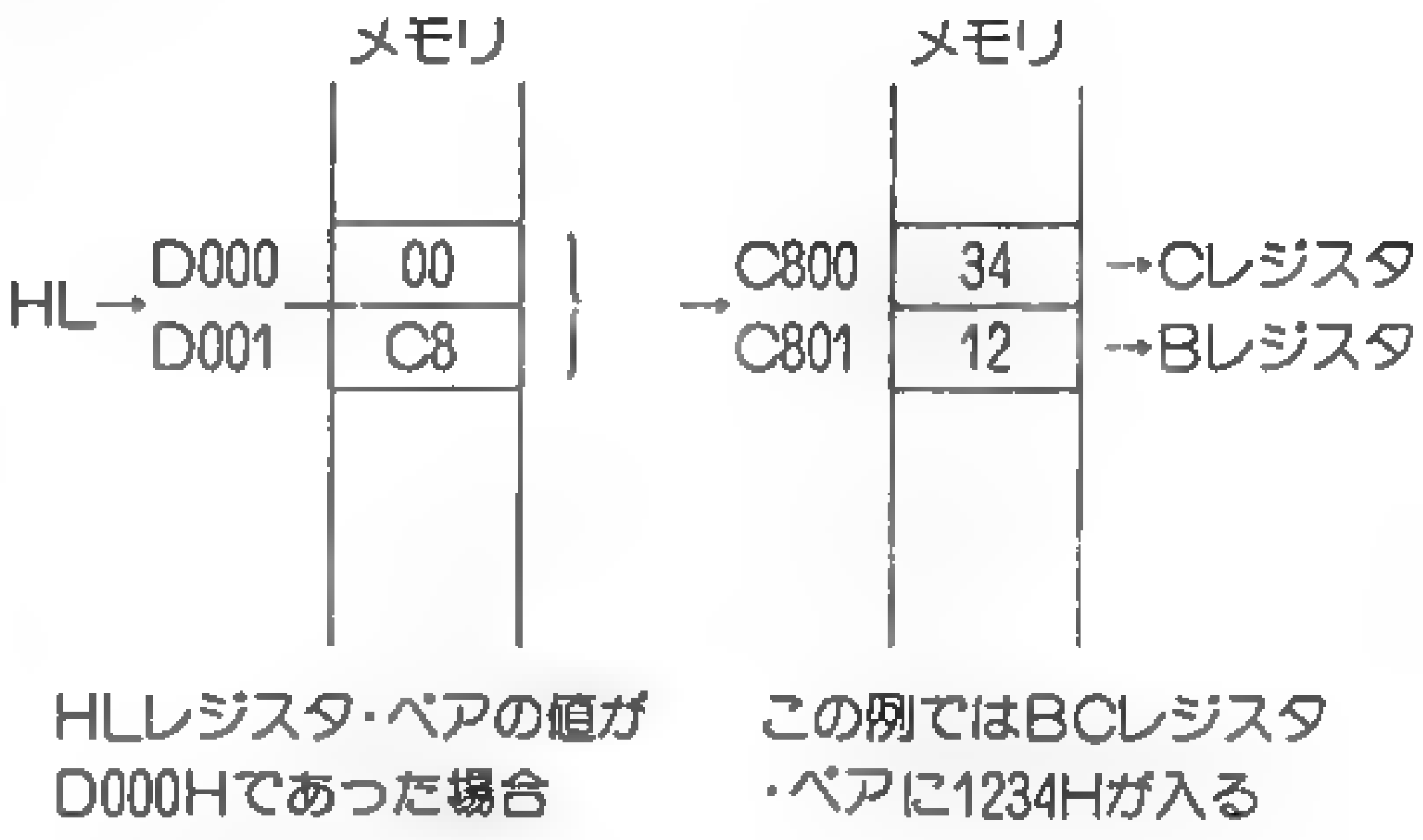
リスト2

```
100 '
110 '   sample call statement
120 '
130 CLEAR ,&HDFFF : DEFINT A-Z
140 MACHINE=&HE000 : WORK=&HE03A : GOSUB *INIT.MACHINE
150 A=10 : B=100 : C=-125 : D=-2560 : E=32767
160 PRINT A,B,C,D,E
170 CALL MACHINE(A,B,C,D,E)
180 FOR I=WORK TO WORK+9 STEP 2
190   J=VAL("&H"+HEX$(PEEK(I+1))+"00")+PEEK(I)
200   PRINT J,
210 NEXT : PRINT
220 END
230 '
240 *INIT.MACHINE
250  FOR I=MACHINE TO WORK-1
260     READ A$
270     POKE I,VAL("&H"+A$)
280  NEXT
290 RETURN
300 DATA D5,CD,29,E0,ED,53,3A,E0,E1,CD,29,E0,ED,53,3C,E0
310 DATA 60,69,2B,CD,2D,E0,ED,43,3E,E0,CD,2D,E0,ED,43,40
320 DATA E0,CD,2D,E0,ED,43,42,E0,C9,5E,23,56,C9,23,CD,29
330 DATA E0,E5,EB,CD,29,E0,42,4B,E1,C9
```

```
                   ;
                   ;   SAMPLE CALL STATEMENT
                   ;
                         ORG   0E000H
                   ;
E000 D5                  PUSH DE            DEレジスタ・ペアを退避する
E001 CD29E0              CALL LOAD1
E004 ED533AE0            LD   (WORK1),DE    DEレジスタの内容(変数Aの値)をWORK1へロードする
E008 E1                  POP  HL           {退避したDEレジスタ・ペアの内容をHLレジスタ・ペアへ
E009 CD29E0              CALL LOAD1        {復帰する
E00C ED533CE0            LD   (WORK2),DE    DEレジスタ・ペアの内容(変数Bの値)をWORK2へロードする
E010 60                  LD   H,B          }BCレジスタ・ペアの内容をHLレジスタ・ペアへロードする
E011 69                  LD   L,C
E012 2B                  DEC  HL           {サブルーチンLOAD2の最初でINC　HLを行うので，そ
E013 CD2DE0              CALL LOAD2        {のためのダミー
E016 ED433EE0            LD   (WORK3),BC   {BCレジスタ・ペアの内容(変数Cの値)をWORK3へロー
E01A CD2DE0              CALL LOAD2        {ドする
E01D ED4340E0            LD   (WORK4),BC   {BCレジスタ・ペアの内容(変数Dの値)をWORK4へロー
E021 CD2DE0              CALL LOAD2        {ドする
E024 ED4342E0            LD   (WORK5),BC   {BCレジスタ・ペアの内容(変数Eの値)をWORK5へロー
                   ;                       {ドする
E028 C9                  RET                BASICへ戻る
                   ;
                   ;
E029 5E            LOAD1: LD   E,(HL)      }HLレジスタ・ペアの示すメモリの内容をEレジスタへ,HL
E02A 23                   INC  HL          }+1の示すメモリの内容をDレジスタへロードする,HLレジ
E02B 56                   LD   D,(HL)      }スタ・ペアはインクリメントされる
E02C C9                   RET
                   ;
                   ;
```

```
E02D 23         LOAD2: INC  HL
E02E CD29E0            CALL LOAD1
E031 E5                PUSH HL
E032 EB                EX   DE,HL
E033 CD29E0            CALL LOAD1
E036 42                LD   B,D
E037 4B                LD   C,E
E038 E1                POP  HL
E039 C9                RET
                ;
                ;     WORK AREA OF VARIABLE
                ;
E03A            WORK1: DS   2  変数Aの値を格納するところ
E03C            WORK2: DS   2  変数Bの値を格納するところ
E03E            WORK3: DS   2  変数Cの値を格納するところ
E040            WORK4: DS   2  変数Dの値を格納するところ
E042            WORK5: DS   2  変数Eの値を格納するところ
                ;
E044                   END
```

HLレジスタ・ペアの示すメモリの内容を下位8ビット，HL＋1の示すメモリの内容を上位8ビットとする．アドレスのメモリの内容をCレジスタにロードて，次のアドレスのメモリの内容をBレジスタにロードするサブルーチン．HLの値は＋2される
下図参照


メモリ
メモリ
HL
D000
D001
00
C8
C800
C801
34
12
→Cレジスタ
→Bレジスタ
HLレジスタ・ペアの値がD000Hであった場合
この例ではBCレジスタ・ペアに1234Hが入る

6.3　内部ルーチンの利用

あるプログラムを作成する際，すでに動作の確認されているプログラムをサブルーチンとして使うことができれば，開発時間を短縮することができます．PC-8801のユーザが簡単に使える既製のプログラムとして，BASICインタプリタ内のルーチンがまず考えられます．例えば，

```
CALL  3E0DH
```

とすれば，Aレジスタの内容のキャラクタを現在カーソルの位置へ簡単に出力することができます．テキスト画面に字を表示するだけなら，テキストVRAMへ直接キャラクタ・コードを書き込めばよいわけですが，この3E0DH番地から始まる内部ルーチンは，実際にはだいぶ複雑な処理を行っており，これと同じ処理を行うルーチンを作成するにはかなり手間がかかりますが，内部ルーチンを使えばたった3バイトで済みます．

このように，非常に便利な内部ルーチンですが，ブラックボックス的に使用するには気を付けなければならない点がいくつかあります．ここでは，この内部ルーチンの使い方についてまとめておきたいと思います．

6.3.1　内部ルーチンとは

まず内部ルーチンとはどういうものかを明確にしておきましょう．

この本で内部ルーチンというのは，私たちユーザが作成したプログラムルーチンではなく，メーカから供給されるプログラムの一部分を指します．主に使用するのは，PC-8801の本体に実装されているROMに格納されたBASICインタプリタ内のルーチンです．

PC-8801には，N_{88}-BASICとN-BASICインタプリタがROMに納められていますが，ここではN_{88}-BASICのルーチンについて説明していきます．

PC-8801は，標準でROM72Kバイト，RAM112Kバイトの合計184Kバイトのメモリを実装しています．ところがCPUμPD780は，0000H番地〜FFFFH番地の64Kバイトしか直接アドレス指定をすることができませんから，バンク切り換えを行ってこれらのメモリをアクセスしています．

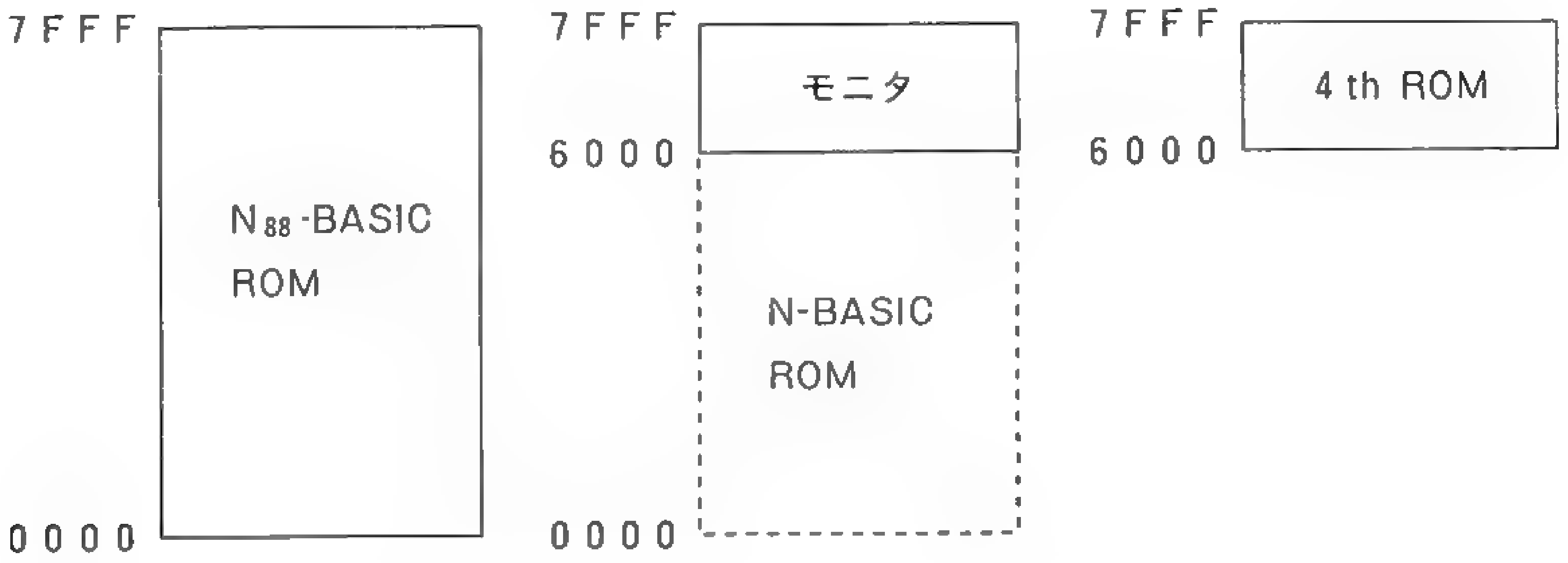


図6.6　N88-BASIC時に使われるROM

N88-BASICが動作しているときには，図６．６に示すROMが使用されます．通常はN88-BASIC ROMが選択されており，必要に応じてモニタのルーチンが入ったN-BASIC ROMやグラフィック関係のルーチンが入っているROM(4th ROM)が選択されています．

アドレス0000H番地〜7FFFH番地の空間には，もう１つ，テキスト・エリアと呼ばれるBASICのソース・プログラムなどが格納されるメモリがありますが，このRAMの取り扱いについては後に述べることにします．

私たちがプログラム中で内部ルーチンを使用する場合には，これらのメモリ・バンクのうちどれが選択されているかということを知らなければなりませんが，これらのバンク切り換えは，I/Oポートのアクセスによって行うことも可能です．

６．３．２　内部ルーチンの使い方

この本のサンプル・プログラムでは，N88-BASIC ROM内のルーチンを使用しています．N88-BASICモードであれば，ROMはすでに選択されていますから，普通のサブルーチンと同じようにCALL命令などを使って簡単に呼び出すことができます．

一般的な使い方を図６．７に示します．

引数は，使用する内部ルーチンによって，任意のレジスタに設定したりメモリ中に設定したりします．

内部ルーチン内でも，様々な用途にレジスタを利用しますから，それが保存されない場合には，内部ルーチンを呼ぶ前にスタックなどへ必要なレジスタを退避させなければなりません．

ではここで，本書で扱う主な内部ルーチンのアドレス，機能，引数，利用されるレジスタについてまとめておきます．

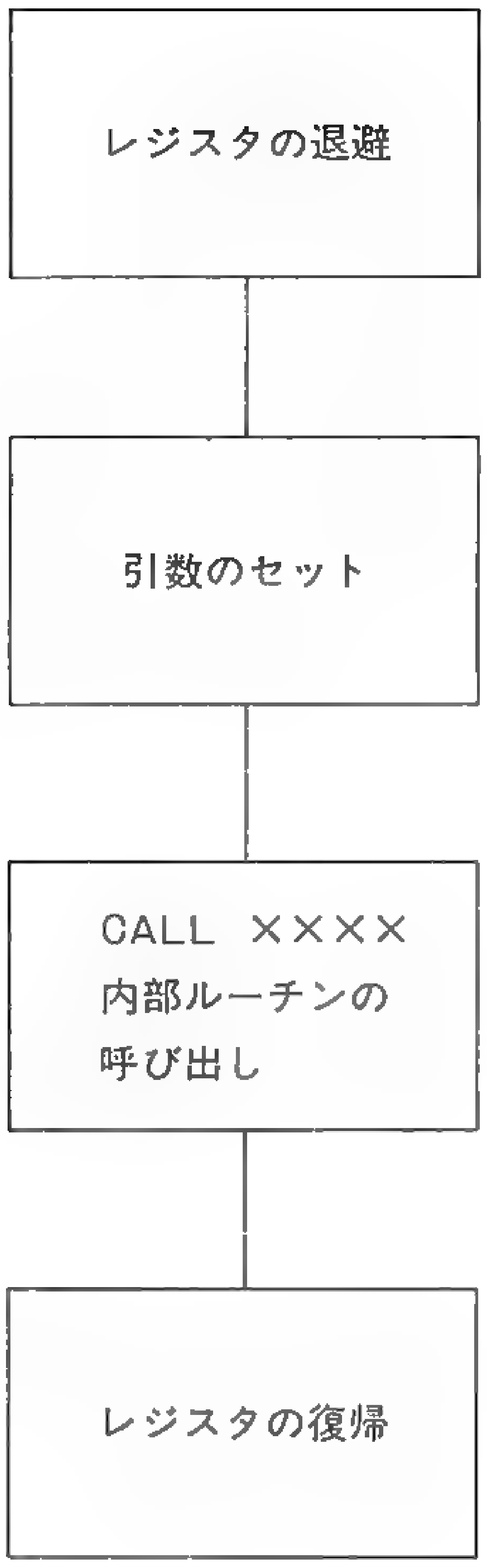

レジスタの退避
引数のセット
CALL ××××
内部ルーチンの
呼び出し
レジスタの復帰

図6.7 内部ルーチンの使い方

■ 1文字入力

アドレス	3583H
機能	キーボードから1文字入力する
説明	キーボードからの入力を待ち，入力があればそのコードをAレジスタに入れて戻る
引数	なし
レジスタ	A，Fが変更される

このルーチンを使用したデータ入力ルーチンを2種類紹介します．

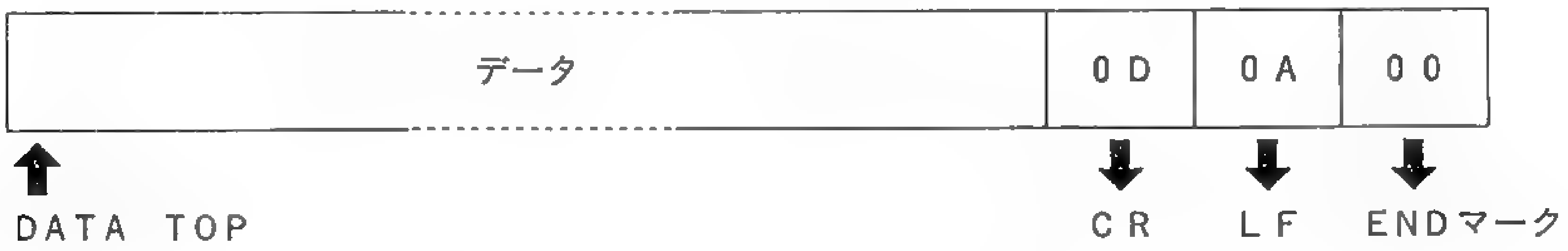

データ
0D
0A
00
DATA TOP
CR
LF
ENDマーク

図6.8 データ・フォーマット(1)

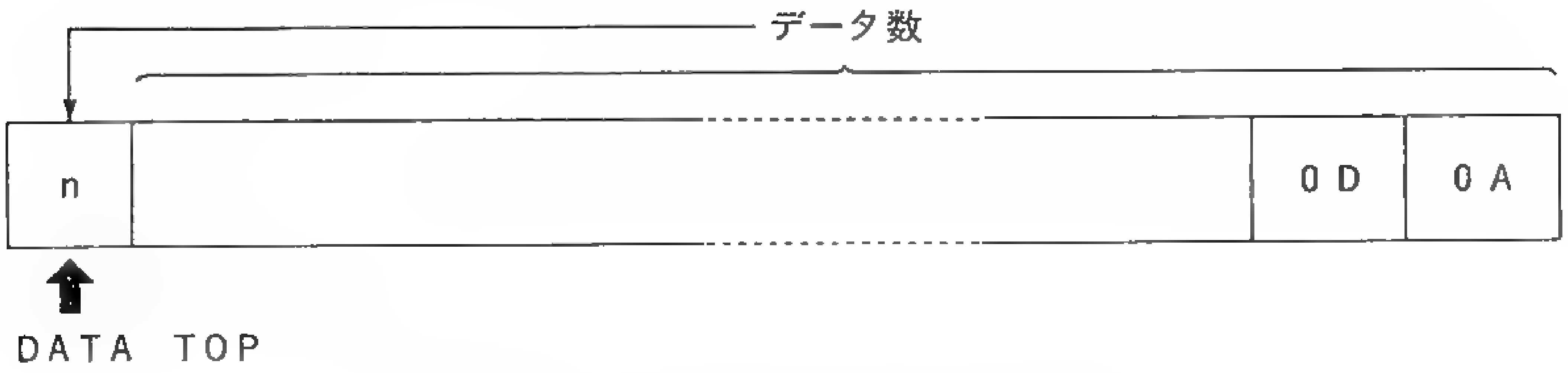

データ数
n
0D
0A
DATA TOP

図6.9 データ・フォーマット(2)

リスト3

```
                    ;          INPUT ROUTINE NO.1
                    ;          2/27     BY M.F
                    ;
BA00                DATTOP:EQU   0BA00H        入力データの格納エリア先頭アドレス
3583                GETCHR:EQU   03583H        キー入力ルーチンのエントリ・アドレス
                    ;
                           ORG   0B900H
B900                START:
B900 CD04B9                CALL  INPUT1        キーボードからデータを取り込み，メモリに格納する
                    ;
B903 FF             EXIT:  RST   38H           モニタへ戻る
                    ;
B904 2100BA         INPUT1:LD    HL,DATTOP     HLに格納エリアの先頭アドレスをセットする
B907 E5                    PUSH  HL
B908 CD8335         INP1:  CALL  GETCHR        1文字入力されるのを待ち，入力データをAに入れる
B90B 77                    LD    (HL),A        データを格納しポインタを進める
B90C 23                    INC   HL
B90D FE0D                  CP    0DH           CRでなければINP1へ飛び，次の入力を行う
B90F 20F7                  JR    NZ,INP1
B911 360A                  LD    (HL),0AH      CRであればLFとENDマーク(0)を付けてリターンする
B913 23                    INC   HL
B914 3600                  LD    (HL),0
B916 E1                    POP   HL
B917 C9                    RET
                    ;
B918                       END
```

リスト4

```
                    ;          INPUT ROUTINE NO.2
                    ;          2/27     BY M.F
                    ;
BA00                DATTOP:EQU   0BA00H        入力データの格納エリア先頭アドレス
3583                GETCHR:EQU   03583H        キー入力ルーチンのエントリ・アドレス
                    ;
                           ORG   0B900H
B900                START:
B900 CD04B9                CALL  INPUT2        キーボードからデータを取り込み，メモリに格納する
                    ;
B903 FF             EXIT:  RST   38H           モニタへ戻る
                    ;
B904 2100BA         INPUT2:LD    HL,DATTOP     データ数を格納するアドレスの保存する
B907 E5                    PUSH  HL
B908 23                    INC   HL
B909 0600                  LD    B,0           データ数は最初0である
B90B CD8335         INP2:  CALL  GETCHR        1文字入力されるのを待ち，入力データをAに入れる
B90E 77                    LD    (HL),A        データを格納しポインタとデータ数をインクリメントする
B90F 23                    INC   HL
B910 04                    INC   B
B911 FE0D                  CP    0DH           CRでなければ次の入力を行う
B913 20F6                  JR    NZ,INP2
B915 360A                  LD    (HL),0AH      CRであれば，LFを付け，データ数をBA00Hに格納してリ
B917 E1                    POP   HL            ターンする
B918 04                    INC   B
B919 70                    LD    (HL),B
B91A C9                    RET
                    ;
B91B                       END
```

リスト3は図6.8のようなフォーマットでデータを取り込むプログラムです．BA00H番地からデータを格納していき，キャリッジリターン(CR：0DH)が入力されるとその次にラインフィード(LF：0AH)，ENDマークとして00Hを続けて格納して処理を終わります．

この例では，入力したデータはCRT上にエコーバックされません．必要があれば，CALL GETCHRの次で出力ルーチンを呼びます．

リスト4は，図6.9のようなフォーマットでデータを取り込むプログラムです．BA00H番地にはデータ数が入り，BA01H番地から実際のデータが格納されていきます．

これらのプログラムは単純にデータを格納していくものですから，実用にするにはもう少し手を加えなければなりません．

このルーチンは，主に1文字のみの入力に使用するとよいでしょう．

■ リアルタイムのキー判定

アドレス	35CEH
機　能	キースキャンを行い，キーが押されていればそのデータを取り込む
説　明	キーが押されているかを調べ，押されていなければZフラグをセット(Z=1)して戻り，押されていればZフラグをリセット(Z=0)し，データをAレジスタに入れて戻る．BASICのINKEY$に似た働きのルーチン
引　数	なし
レジスタ	A，Fが変更される

このルーチンは，リアルタイムで入力を調べる時に使用します．リスト5では，CRTへの出力を，CTRL-S(13H)の入力で一時停止して，CTRL-C(STOP：03H)の入力で処理を終了しています．この例は，入力待ちを行うルーチンとそうでない入力ルーチンをうまく利用したもので，一時停止しているときはCTRL-Cの入力で終了，他のキーの入力で処理を再開します．

リスト5

```
                  ;        INKEY TEST
                  ;        3/14     BY M.F
                  ;
35CE              INKEY: EQU   35CEH     リアルタイムなキー入力ルーチンのエントリ・アドレス
3583              GETCHR:EQU   3583H     キー入力ルーチンのエントリ・アドレス
3E0D              PUTCHR:EQU   3E0DH     CRT出力ルーチンのエントリ・アドレス
                  ;
                         ORG   0B900H
B900              START:
B900 CD12B9              CALL  DISP      CRTに'M'を出力する
B903 CDCE35              CALL  INKEY     キーボードをスキャンしてリアルタイムなキー入力を行う
B906 28F8                JR    Z,START   キー入力がなければSTARTへ飛ぶ
B908 FE13                CP    13H       CTRL-Sが押されていれば，GETCHRをコールして，何らか
B90A CC8335              CALL  Z,GETCHR  のキーが押されるまで待つ
B90D FE03                CP    03H       CTRL-Cが押されていれば，EXITでモニタに戻る．それ以外
B90F 20EF                JR    NZ,START  であればSTARTへ飛び，ループになる
```

```
                     ;
B911 FF              EXIT:  RST  38H
                     ;
B912 3E4D            DISP:  LD   A,'M'    ⎫
B914 CD0D3E                 CALL PUTCHR   ⎬CRTに'M'を出力するルーチン
B917 C9                     RET           ⎭
                     ;
B918                        END
```

■ ライン入力

アドレス	5F92H
機　　能	ライン入力を行う
説　　明	このルーチンは，スクリーン・エディット機能をもったライン入力を行う．3583H番地や35CEH番地の入力ルーチンと異なり，入力した文字をCRTにエコーバックする．データはE9B9H番地からのバッファに格納され，データの最後にはENDマークとして00Hが書かれる．バッファはE9B9H番地～EAB7H番地の255バイトだが，最後は必ず00Hでなければならないから，実際には254バイトのデータを入力することができる．254バイトを超えて入力した場合は，先頭から254バイトが有効とされる キャリッジリターン(0DH)，CTRL-C(03H)の入力でこのルーチンから戻るが，CTRL-Cを入力した場合には中断したとしてバッファには何も入力されず，CYフラグがセット(CY=1)される HLレジスタ・ペアにE9B8Hが，Aレジスタには最後に入力したデータが格納されて戻る
引　　数	なし
レジスタ	A，F，B，C，D，E，H，Lが変更される

リスト6は，このルーチンで入力したデータ全体を表示するプログラムです．キャリッジリターン(0DH)の入力によりバッファの内容を表示して再び入力を待ちます．CTRL-C(03H)が入力されると，モニタへ戻ります．このルーチンはスクリーン・エディット機能が使えますから，各種コントロール・キーによるスクリーン制御や，カーソル移動を試してみるとよいでしょう．

リスト6

```
                     ;        TEST LINE INPUT
                     ;        3/14    BY M.F
                     ;
5F92                 LINPT1:EQU  5F92H      ライン入力ルーチンのエントリ・アドレス
3E0D                 PUTCHR:EQU  3E0DH      CRT出力ルーチンのエントリ・アドレス
                     ;
                            ORG  0B900H
B900                 START:
B900 CD12B9                 CALL CRLF       改行する
B903 CD925F                 CALL LINPT1     スクリーン・エディットでライン入力する．入力データはE9B9Hから格納される
B906 3815                   JR   C,EXIT     STOPキーでライン入力を中断した場合はEXIT へ飛ぶ
```

```
                   ;
B908 23            L0:     INC   HL          バッファのポインタのインクリメント（最初 ┐
                                             は，バッファのスタート・アドレスを指す） │
B909 7E                    LD    A,(HL)      バッファから1文字取り出す               │ バッファの内容を表示
B90A B7                    OR    A           ┐ENDマークの0であれば表示せずに次の     ├ する
B90B 28F3                  JR    Z,START     ┘入力のためSTARTへ飛ぶ                  │
B90D CD0D3E                CALL  PUTCHR      CRTへ1文字出力する                      ┘
B910 18F6                  JR    L0
                   ;
B912 3E0D          CRLF:   LD    A,0DH     ┐
B914 CD0D3E                CALL  PUTCHR    │
B917 3E0A                  LD    A,0AH     ├ CRとLFを出力して改行する
B919 CD0D3E                CALL  PUTCHR    │
B91C C9                    RET             ┘
                   ;
B91D FF            EXIT:   RST   38H         モニタへ戻る
                   ;
B91E                       END
```

■ CRT出力

アドレス	3E0DH
機　　能	CRTに1文字出力する
説　　明	Aレジスタの内容をテキスト画面のカーソルの位置に出力し，カーソルの位置を進める．このルーチンは，メッセージなど文字を画面に表示するときにテキストVRAMのアドレスはいっさい考える必要がなく，いわゆるCRTターミナルへデータを送る場合と同じように扱うことができる．データがコントロール・コードであれば，それに応じた画面制御，例えば CTRL -L(0CH)の出力によりテキスト画面のクリアと，カーソルのホームポジションへの移動を行う
引　　数	Aレジスタに出力するデータをセットする
レジスタ	すべて保存される

このルーチンは1文字の出力ですから，複数の文字すなわちメッセージの出力プログラムを紹介します．

リスト7

```
                    ;        MESSAGE OUTPUT ROUTINE1
                    ;        12/24    BY M.F
                    ;
3583                GETCHR:EQU  3583H     キー入力ルーチンのエントリ・アドレス
3E0D                PUTCHR:EQU  3E0DH     CRT出力ルーチンのエントリ・アドレス
                    ;
                           ORG  0B900H
B900                START:
B900 CD1AB9                CALL MESG1C    改行をしてから，文字列(MACHINE)を出力する．B90AHにリターンする
B903 4D414348              DB   'MACHINE',0 文字列を設定する．最後の0は文字列の終わりを示すENDマ
B907 494E4500                             ーク
                    ;                                         ┐改行して，文字列（FRIEND !）を出力
B90B CD1DB9                CALL MESG1                         ├する．データとしてCR，LFのコードを設
B90E 0D0A4652              DB   0DH,0AH,'FRIEND !',0          ┘定している
B912 49454E44
B916 202100
```

```
                   ;
B919 FF            EXIT:  RST   38H        モニタへ戻る
                   ;
B91A CD28B9        MESG1C:CALL  CRLF       改行をする
B91D E3            MESG1: EX    (SP),HL    スタックから,本来の戻り番地をHLに取り出す,これは文字列のポイ
                                           ンタとなる
B91E 7E                   LD    A,(HL)     )
B91F B7                   OR    A          } データを取り出し,ENDマークをチェックし,ポインタを次に進める
B920 23                   INC   HL         )
B921 E3                   EX    (SP),HL    スタックへ戻り番地をもどす. ポインタとしても使用されるから, も
                                           との戻り番地ではない
B922 C8                   RET   Z          B91FHのOR Aのフラグ変化をみている.0であればリターンする
B923 CD0D3E               CALL  PUTCHR     CRTに出力する
B926 18F5                 JR    MESG1      MESG1へ飛びループになる
                   ;
B928 3E0D          CRLF:  LD    A,0DH      )
B92A CD0D3E               CALL  PUTCHR     |
B92D 3E0A                 LD    A,0AH      } CR, LFのコードを出力して改行するルーチン
B92F CD0D3E               CALL  PUTCHR     |
B932 C9                   RET              )
                   ;
B933                      END
```

リスト8

```
                   ;         MESSAGE OUTPUT ROUTINE2
                   ;         12/24   BY M.F
                   ;
3583               GETCHR:EQU   3583H      キー入力ルーチンのエントリ・アドレス
3E0D               PUTCHR:EQU   3E0DH      CRT出力ルーチンのエントリ・アドレス
                   ;
                          ORG   0B900H
B900               START:
B900 210DB9               LD    HL,STR1    ) HLに出力する文字列の先頭アドレスをセットする
B903 CD20B9               CALL  MESG2C     ) 改行した後ENDマーク(0)までのデータを出力する
                   ;
B906 2115B9               LD    HL,STR2    )
B909 CD23B9               CALL  MESG2      )               〃
                   ;
B90C FF            EXIT:  RST   38H        モニタへ戻る
                   ;
B90D 4D414348 STR1:       DB    'MACHINE',00              ) 文字列データを設定する
B911 494E4500                                             } 最後の0は文字列の終りを示すENDマ
                   ;                                      | ーク
B915 0A0D4652 STR2:       DB    0AH,0DH,'FRIEND !',00     )
B919 49454E44
B91D 202100
                   ;
B920 CD2CB9        MESG2C:CALL  CRLF       改行する
B923 7E            MESG2: LD    A,(HL)     )
B924 B7                   OR    A          } 文字列データを取り出す. ENDマーク(0)であればリターンする
B925 C8                   RET   Z          )
B926 CD0D3E               CALL  PUTCHR     CRTに出力する
B929 23                   INC   HL         ポインタを進める
B92A 18F7                 JR    MESG2      MESG2へ飛びループになる
                   ;
B92C 3E0A          CRLF:  LD    A,0AH      )
B92E CD0D3E               CALL  PUTCHR     |
B931 3E0D                 LD    A,0DH      } CR, LFのコードを出力して改行するルーチン
B933 CD0D3E               CALL  PUTCHR     |
B936 C9                   RET              )
                   ;
B937                      END
```

リスト７とリスト８は，図６．８のフォーマットで用意されたデータを表示するプログラムです．

MESG1の方はCALL命令の次に文字列を設定します．MESG 2 は，CALLする前にHLレジスタ・ペアに文字列の先頭アドレスを設定します．これは好みの問題かもしれませんが，MESG1の方が，出力するメッセージがどのような内容であるかがすぐわかりますから便利であるように思われます．

MESG1は，CALL命令によってスタックに積まれるリターン・アドレスを取り出し，それをポインタとしてデータを出力し，文字列のENDマークである00Hがあると処理を終了します．MESG1では，H，L各レジスタは保存され，ENDマークの次のアドレスへ戻ります．

■プリンタ出力

アドレス	3ED4H
機　能	プリンタへ１文字出力する
説　明	Aレジスタの内容をプリンタへ出力する．出力前にキー入力を調べ，CTRL -C(03H) の入力があればプリンタに出力しないで戻る
引　数	Aレジスタに出力するデータをセットする
レジスタ	すべて保存される

リスト9

```
                 ;         TEST PRNTER OUTPUT
                 ;         3/15     BY M.F
                 ;
3ED4             OUTPRN:EQU   3ED4H      プリンタ出力ルーチンのエントリ・アドレス
3E0D             PUTCHR:EQU   3E0DH      CRT出力ルーチンのエントリ・アドレス
                 ;
                        ORG   0B900H
B900             START:
B900 060A               LD    B,10       10行出力するためのカウンタ値をセットする
B902 3E41        L1:    LD    A,'A'     ┐
B904 CD19B9      L0:    CALL  OUT1      │
B907 3C                 INC   A         ├'A'から'Z'までの26文字を出力する
B908 FE5B               CP    'Z'+1     │
B90A 38F8               JR    C,L0      ┘
                 ;
B90C 3E0D               LD    A,0DH     ┐
B90E CD19B9             CALL  OUT1      ├CRコードとLFコードを出力して改行する
B911 3E0A               LD    A,0AH     │
B913 CD19B9             CALL  OUT1      ┘
                 ;
B916 10EA               DJNZ  L1         10行出力したらループを抜ける
                 ;
B918 FF          EXIT:  RST   38H        モニタへ戻る
                 ;
B919 CDD43E      OUT1:  CALL  OUTPRN    ┐
B91C CD0D3E             CALL  PUTCHR    ├プリンタとCRTにAレジスタの内容を出力する
B91F C9                 RET             ┘
                 ;
B920                    END
```

リスト9は，AからZまでの文字を改行しながらプリンタとCRTに10回出力するプログラムです．

途中で CTRL -C(03H)の入力があれば，以後プリンタには出力せず，CRTにだけ出力していきます．これは，03Hの入力があったことを示すフラグがそのままになるため，毎回OUTPRNで03Hの入力があったものと判断されるためです．それによって，プリンタが使用できない場合に CTRL -Cまたは STOP の一度の入力でプログラムの停止から抜け出ることができます．

6.4　データの型との変換

6.4.1　1バイトの表すもの

ここに01000001というビット列で表される1バイトのデータがあるとします．プログラム内にこのデータが出てきたときそれをどのように解釈するかは，プログラムを作成する人が判断しなければなりません．

例えばこれをアスキー・コードとするならば，大文字の〝A〞を表します．ところが，8ビットの数値と考えるならば，これは10進数で65の大きさの数と解釈できます．また，BCDの表現では41という数を表します．

11111110というデータの場合はどうでしょう．BCDでは，4ビット単位で，しかも0000～1001でなければなりませんからこのようなデータは存在しません．単純に2進→10進変換を行うと254という値になりますが，これはデータを8ビットの符号なし数値とみなした場合で，符号付き(2の補数)とするならば－2となります．

このように，メモリに記憶されているデータは単なる0と1の集まりですから，それを扱いやすいデータの型に変換することが必要になることがあります．この節では，実際にマシン語プログラム中でよく使われる各種データ変換について説明します．扱う数値データは符号なしのものです．

6.4.2　データ変換(1)

アスキー・コードで表現された2進数，10進数，16進数を，数値データに変換するプログラムを考えます．これは，キーボードからの入力値(内部ルーチンGETCHR：3583Hなどを使用したとき)を演算に適したデータにする場合などに必要となります．

■ ASC(BIN) TO BIN

まず，アスキー・コードで表した2進数を1バイトの2進データに変換するプログラムをリスト10に示します．2進数は0と1で表しますから，アスキー・コードでは30Hと31Hの2つのデータを使います．そしてこの30Hと31Hのデータ列がそのままのイメージで各ビットに対応します．

リスト10

```
                    ;       ASC(BIN) TO BIN
                    ;       2/7      BY M.F
                    ;
3583                GETCHR:EQU   3583H       キー入力ルーチンのエントリ・アドレス
3E0D                PUTCHR:EQU   3E0DH       CRT出力ルーチンのエントリ・アドレス
                    ;
                           ORG   0B900H
B900                ASBIBX:
B900 2100BA                LD    HL,0BA00H   変換データの格納エリア先頭アドレスをHLにセットする
B903 CD25B9         ABBX1: CALL  CRLF        改行する
B906 0E00                  LD    C,0         Cにデータが生成されるから，クリアしておく
B908 CD8335         ABBX2: CALL  GETCHR      ┐
B90B FE0D                  CP    0DH         │
B90D 2812                  JR    Z,WRITE     │キー入力をして，リターンであればWRITEへ飛ぶ
B90F FE30                  CP    30H         │'0'と'1'であればABBX3へ飛ぶ
B911 2804                  JR    Z,ABBX3     │
B913 FE31                  CP    31H         ┘
B915 2019                  JR    NZ,EXIT      30Hと31HでなければEXITへ飛ぶ
B917 CD0D3E         ABBX3: CALL  PUTCHR      入力データを表示する
B91A FE31                  CP    31H         } 30HならCY=0，31HならCY=1にする
B91C 3F                    CCF
B91D CB11                  RL    C           CYと共にデータを左から詰めていき，変換する
B91F 18E7                  JR    ABBX2       次の桁の入力のためABBX2へ飛ぶ
                    ;
B921 71             WRITE: LD    (HL),C      }Cに生成された変換データを格納エリアに書き，ポインタを進
B922 23                    INC   HL          }める
B923 18DE                  JR    ABBX1       新たな入力のためABBX1へ飛ぶ
                    ;
B925 3E0D           CRLF:  LD    A,0DH       ┐
B927 CD0D3E                CALL  PUTCHR      │
B92A 3E0A                  LD    A,0AH       │CRとLFを出力して改行をするルーチン
B92C CD0D3E                CALL  PUTCHR      │
B92F C9                    RET               ┘
                    ;
B930 FF             EXIT:  RST   38H         モニタへ戻る
                    ;
B931                       END
```

入力データ	30H，31H，30H，31H，31H，30H，31H	8バイト
変換結果	0　1　0　1　1　0　1	1バイト

図6.10　データの対応(1)

この例では，RETURN が押されるまでキーボードからデータを取り込み，変換した結果をBA00H番地から格納していきます．0，1 の入力であれば，それをCYフラグの0，1に対応させて，Cレジスタの内容をLSBからMSBの方向にシフトすることによってデータを変換していきます．

1バイトは8ビットですから，8個のアスキー・コードのデータが必要ですが，入力ルーチンをリストのようにすることによって，データが8個より多い場合には後から数えて8個のデータが有効となり，少ない場合にも正しく変換することができます．

0, 1, RETURN 以外のキーを入力したときには，結果を格納せずにプログラムを終了して，モニタのコマンド・レベルに戻ります．その結果はDコマンドなどで確認することができます．

■ ASC(HEX) TO BIN

次にアスキー・コードで表した16進数2文字を1バイトの2進データに変換するプログラムをリスト11に示します．

リスト11

```
                ;       ASC(HEX) TO BIN
                ;       2/7     BY M.F
                ;
3583            GETCHR:EQU  3583H      キー入力ルーチンのエントリ・アドレス
3E0D            PUTCHR:EQU  3E0DH      CRT出力ルーチンのエントリ・アドレス
                ;
                       ORG  0B900H
B900            ASBIHX:
B900 2100BA            LD   HL,0BA00H  変換データの格納エリア先頭アドレスをHLにセットする
B903 CD36B9     ABHX1: CALL CRLF       改行をする
B906 0E00              LD   C,0        Cにデータが生成されるからクリアしておく
B908 CD8335     ABHX3: CALL GETCHR     ┐
B90B FE0D              CP   0DH        }キー入力を行い，リターンであればWRITEへ飛ぶ
B90D 2823              JR   Z,WRITE    ┘
B90F CB21              SLA  C          ┐
B911 CB21              SLA  C          }Cの下位4ビットを上位4ビットとし，下位4ビットは0になる
B913 CB21              SLA  C          │
B915 CB21              SLA  C          ┘
B917 47                LD   B,A        入力データをBに保存する
B918 FE47              CP   'F'+1      }アスキー・コードの47H以上であればEXITへ飛ぶ
B91A 3024              JR   NC,EXIT    ┘
B91C D630              SUB  30H        }データから30Hを引く
B91E 3820              JR   C,EXIT     }入力データが0～2FHであったならEXITへ飛ぶ
B920 FE0A              CP   0AH        }入力データが30H～39HであればABHX2へ飛ぶ
B922 3806              JR   C,ABHX2    ┘
B924 FE11              CP   11H        }入力データが3AH～40HであればEXITへ飛ぶ
B926 3818              JR   C,EXIT     ┘
B928 D607              SUB  7          入力データが41H～46Hであればさらに7を引く．つまりこれで37
                                       Hを引いたことになる
                ;
B92A B1         ABHX2: OR   C          }Cの下位ビットに変換データ（0～0FH）を書く
B92B 4F                LD   C,A        ┘
B92C 78                LD   A,B        }入力データを表示する
B92D CD0D3E            CALL PUTCHR     ┘
B930 18D6              JR   ABHX3      次の桁の入力のためABHX3へ飛ぶ
                ;
B932 71         WRITE: LD   (HL),C     }Cに生成されたデータを格納エリアに書き，ポインタを進める
B933 23                INC  HL         ┘
B934 18CD              JR   ABHX1      新たな入力のためABHX1へ飛ぶ
                ;
B936 3E0D       CRLF:  LD   A,0DH      ┐
B938 CD0D3E            CALL PUTCHR     │
B93B 3E0A              LD   A,0AH      }CRとLFを出力して改行する
B93D C30D3E            JP   PUTCHR     │
                ;*      RET            ┘
                ;
B940 FF         EXIT:  RST  38H        モニタへ戻る
                ;
B941                   END
```

入力データ	3	B	2バイト
変換結果	0 0 1 1	1 0 1 1	1バイト

図6.11 データの対応(2)

16進数は，0～9，A～Fの16種類のキャラクタで表現されますが，これはアスキー・コードの30H～39H，41H～46H(小文字は考えない)に相当します．入力データの1バイト，つまり16進数の1桁は，結果として4ビットの2進データに対応しています．

この変換には，簡単な減算を使います．30H～39Hまでのデータは，30Hを引くことによって0000～1001の4ビット・データを簡単に得ることができます．41Hから46Hまでのデータは，37Hを引くことによって1010～1111のデータを得ることができます．実際には，16進数のキャラクタであるか否かをチェックしながらまず30Hを引き，その後これがA～Fであった場合のみ7を引いています．入力および結果の格納方法はリスト10と変わりません．

■ ASC(DEC) TO BIN

2進や16進のように，2のべき乗を基数とするデータの変換は比較的容易に作ることができるのですが，アスキー・コードの10進数を2進データに変換するには少し考えなければなりません．リスト12は，5桁のアスキー・コードで表された10進数を2バイトの2進コードに変換するプログラムです．

リスト12

```
                  ;       ASC(DEC) TO BIN
                  ;       3/28    BY M.F
                  ;
3E0D              PUTCHR:EQU  3E0DH       CRT出力ルーチンのエントリ・アドレス
3583              GETCHR:EQU  3583H       キー入力ルーチンのエントリ・アドレス
0005              BSIZE: EQU  5           入力バッファのサイズ，5桁まで
                  ;
                         ORG  0B900H
B900              ASBIDX:
B900 DD2100BA            LD   IX,0BA00H   IXに結果の格納エリア先頭アドレスをセットする
B904 CD31B9       ASDX2: CALL DECIN       10進数5桁の入力を行う.データは30H引かれてINBUFに格納される
B907 FE0D                CP   0DH        ┐
B909 2801                JR   Z,ASDX1    │
                  ;                      ├DECINがCRで終了していなければモニタへ戻る
B90B FF           EXIT:  RST  38H        │
                  ;                      ┘
B90C 0605         ASDX1: LD   B,BSIZE     Bに桁数である5を設定する
B90E FD2168B9            LD   IY,INBUF    IYに入力バッファの先頭アドレスをセットする
B912 210000              LD   HL,0
                  ;
```

```
B915 29         ASDEC1:ADD  HL,HL
B916 5D                LD   E,L
B917 54                LD   D,H          HLのデータを10倍する
B918 29                ADD  HL,HL                                  INBUFの5桁のデータに
B919 29                ADD  HL,HL                                  10進数の重みをつけて加
B91A 19                ADD  HL,DE                                  算して変換を行う
                ;                                                  結果はHLに生成される
B91B FD5E00            LD   E,(IY)
B91E 1600              LD   D,0          INBUFからデータを取り込む
B920 19                ADD  HL,DE
B921 FD23              INC  IY
B923 10F0              DJNZ ASDEC1
                ;
B925 DD7500            LD   (IX),L
B928 DD7401            LD   (IX+1),H     変換データを格納しポインタを進める
B92B DD23              INC  IX
B92D DD23              INC  IX
B92F 18D3              JR   ASDX2        新たな入力のためASDX2へ飛ぶ
                ;
B931 CD5DB9     DECIN: CALL CRLF
B934 2168B9            LD   HL,INBUF
B937 0605              LD   B,BSIZE
B939 3600       DECIN1:LD   (HL),0       INBUFのイニシャライズ（クリア）
B93B 23                INC  HL
B93C 10FB              DJNZ DECIN1
                ;
B93E CD8335     DECIN2:CALL GETCHR
B941 FE0D              CP   0DH
B943 C8                RET  Z
B944 FE3A              CP   '9'+1        キー入力を行い,CRか'0'～'9'以外の入力であればリターン
B946 D0                RET  NC           する
B947 FE30              CP   30H          '0'～'9'のデータであればCRTに出力して以下へ
B949 D8                RET  C
B94A CD0D3E            CALL PUTCHR
B94D D630              SUB  30H          30H～39Hのデータを0～9にする
                ;
B94F 2169B9            LD   HL,INBUF+1
B952 1168B9            LD   DE,INBUF
B955 010400            LD   BC,BSIZE-1   INBUFに入力されたデータを格納していく．最後に入力さ
B958 EDB0              LDIR              れた5バイトが有効となるようにデータをずらしていく
B95A 12                LD   (DE),A
B95B 18E1              JR   DECIN2
                ;
B95D 3E0D       CRLF:  LD   A,0DH
B95F CD0D3E            CALL PUTCHR
B962 3E0A              LD   A,0AH        改行をする
B964 CD0D3E            CALL PUTCHR
B967 C9                RET
                ;
B968            INBUF: DS   BSIZE        入力バッファとして5バイト確保する
                ;
B96D                   END
```

10進数の各桁は10のべき乗の重みをもっています．n桁目のデータをDnとすると，この場合は５桁ですから

$$D_5\times10^4+D_4\times10^3+D_3\times10^2+D_2\times10+D_1\times1$$

という計算を行えば2進のデータを得ることができます．リスト12では，この式を

$$(((10\times D_5+D_4)\times 10+D_3)\times 10\times D_2)\times 10+D_1$$

のように考えてプログラムしたものです．

ここで，この式をみれば，数の重みを作るために10倍をするルーチンが必要になることがわかります．乗算においては，2のべき乗倍することは比較的容易に行うことができるため，10倍を8倍（2^3倍）したものと，2倍したものを加えることによって実現しています．

入力は，DECINで行っています．DECINでは，5バイトのデータをINBUFに，後から入力したものが有効となるように取り込むと共に，30Hを引いています．

なおこのプログラムでは，5桁の10進数で65535までの入力を正確に変換できます．65536以上の数はオーバフローとなりますが，無視しているため再び0からのデータになってしまいます．結果は，BA00H番地から2バイト単位で格納されていきます．

■ ASC(DEC) TO BCD

次に，アスキー・コードの10進数をBCDに変換するプログラムをリスト13に示します．BCDは前に説明した通り4ビットの0000～1001で10進数の0～9を表すものですから，アスキー・コードの2バイトがBCDの1バイトになります．このリスト13では，DECINで入力したデータを変換して，BA00H番地から格納していきます．

ここで，RLDという命令が出てきますので少し説明しておきます．

RLDはローテイト命令の一種ですが，これまでに紹介したローテイト命令とだいぶ動作が異なります．この命令は，HLレジスタ・ペアの示すアドレスの内容とAレジスタの下位4ビットの間を，4ビット単位でローテイトしていきます．同様に逆向きにローテイトする命令がRRDです．

これらの命令は4ビット単位で動作するため，BCDに関するプログラムで使用されることがあります．

リスト13

```
                  ;        ASC TO BCD
                  ;        3/28    BY M.F
                  ;
3E0D              PUTCHR:EQU   3E0DH         CRT出力ルーチンのエントリ・アドレス
3583              GETCHR:EQU   3583H         キー入力ルーチンのエントリ・アドレス
0005              BSIZE: EQU   5             入力バッファのサイズ
BA00              ODATA: EQU   0BA00H        結果の格納エリアの先頭アドレス
                  ;
                         ORG   0B900H
B900              START:
B900 CD2BB9              CALL  DECIN         10進5桁の入力を行う．データは30H引かれてINBUFに格納される
B903 FE0D                CP    0DH           } DECINがCRで終了していなければEXITへ飛ぶ
B905 200B                JR    NZ,EXIT
```

```
                ;
B907 0605               LD    B,BSIZE      Bにデータ数をカウンタとして設定する(0は注意)
B909 1162B9             LD    DE,INBUF     DEに入力バッファの先頭アドレスをセットする
B90C 2100BA             LD    HL,ODATA     HLに結果格納エリアの先頭アドレスセットする
B90F CD13B9             CALL  ASCBCD       Bに設定したデータ数のアスキー・コードデータをBCDに変換する
                ;
B912 FF         EXIT:   RST   38H          モニタへ戻る
                ;
B913 3600       ASCBCD:LD     (HL),0
B915 CD25B9             CALL  ASBC1
B918 2807               JR    Z,ASBC2      結果の格納エリアをクリアし，INBUFの2バイトのそれぞ
B91A CD25B9             CALL  ASBC1        れ下位4ビットを合わせて，1バイトのBCDとして格納して
B91D C8                 RET   Z            いく
B91E 23                 INC   HL
B91F 18F2               JR    ASCBCD
                ;
B921 AF         ASBC2:  XOR   A
B922 ED6F               RLD                入力データ数が奇数であった場合，最後のデータを詰める
B924 C9                 RET
B925 1A         ASBC1:  LD    A,(DE)
B926 ED6F               RLD
B928 13                 INC   DE           INBUFからデータをAに取り込み，RLDで下位4ビット
B929 05                 DEC   B            を格納する
B92A C9                 RET
                ;
B92B CD57B9     DECIN:  CALL  CRLF         リスト12を参照
B92E 2162B9             LD    HL,INBUF
B931 0605               LD    B,BSIZE
B933 3600       DECIN1:LD     (HL),0
B935 23                 INC   HL
B936 10FB               DJNZ  DECIN1
                ;
B938 CD8335     DECIN2:CALL   GETCHR
B93B FE0D               CP    0DH
B93D C8                 RET   Z
B93E FE3A               CP    ´9´+1
B940 D0                 RET   NC
B941 FE30               CP    30H
B943 D8                 RET   C
B944 CD0D3E             CALL  PUTCHR
B947 D630               SUB   30H
                ;
B949 2163B9             LD    HL,INBUF+1
B94C 1162B9             LD    DE,INBUF
B94F 010400             LD    BC,BSIZE-1
B952 EDB0               LDIR
B954 12                 LD    (DE),A
B955 18E1               JR    DECIN2
                ;
B957 3E0D       CRLF:   LD    A,0DH
B959 CD0D3E             CALL  PUTCHR
B95C 3E0A               LD    A,0AH
B95E CD0D3E             CALL  PUTCHR
B961 C9                 RET
                ;
B962            INBUF:  DS    BSIZE
                ;
B967                    END
```

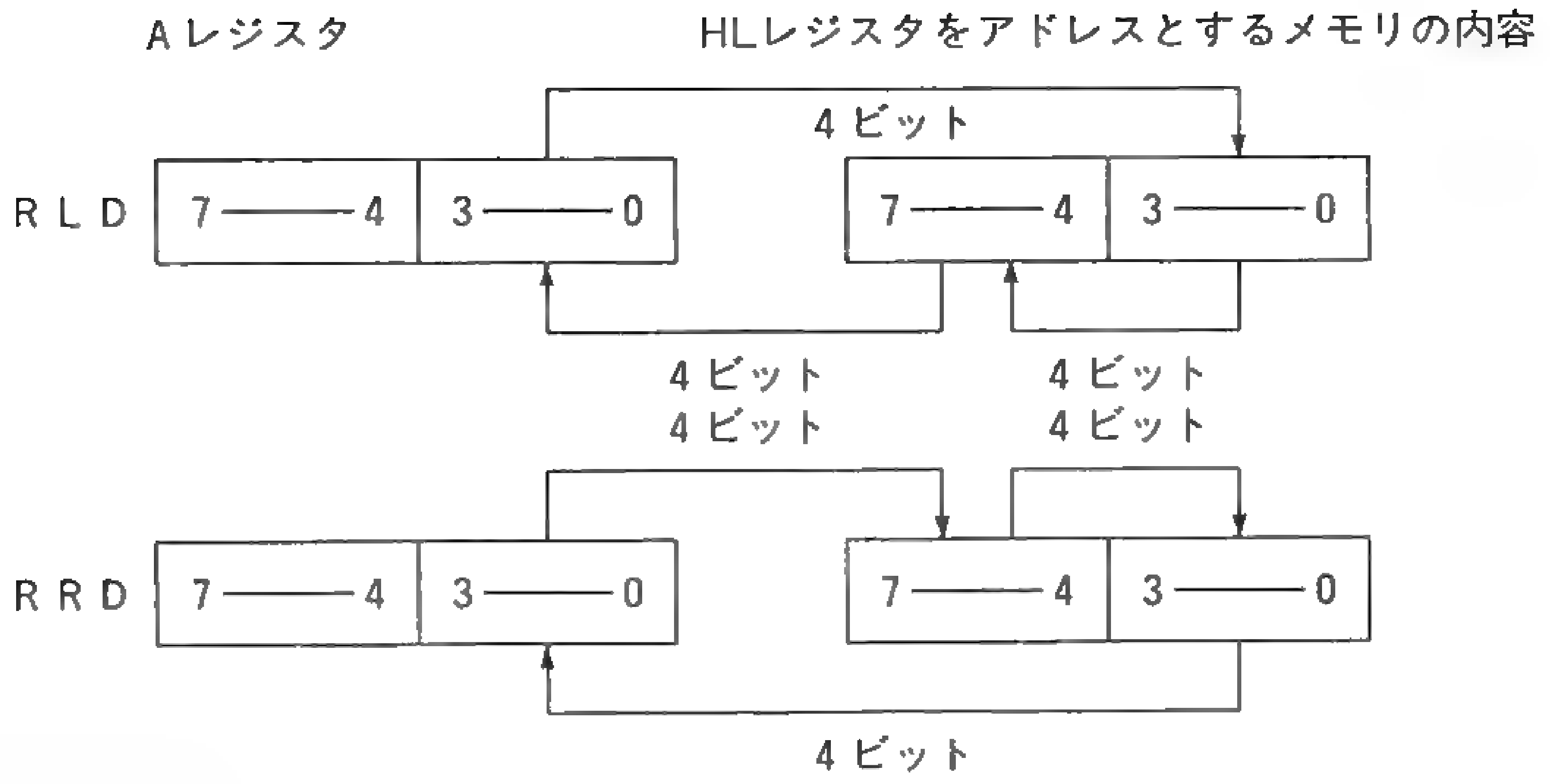

Aレジスタ
HLレジスタをアドレスとするメモリの内容
4ビット
RLD
7——4
3——0
7——4
3——0
4ビット
4ビット
4ビット
4ビット
RRD
7——4
3——0
7——4
3——0
4ビット

図6.12 RLDとRRD

■ 大文字と小文字の変換

入力において，アルファベットの大文字，小文字のどちらでも動作するようにしたいときがあります．その場合には，文字を判断する前に大文字を小文字に(またはその逆)変換しておけば，2回較べる手間が省けます．

アスキー・コードにおいて，大文字のAからZは41Hから5AHに，小文字のaからzは61H~7AHに，それぞれ対応しています．また大文字と小文字の違いは，ビット5が1であるか0であるかということです．

a : 61H 01100001

A : 41H 01000001

↑

ビット5

図6.13 小文字と大文字の違い

ですから，大文字，小文字の変換はこのビット5を操作することによって簡単に行えます．しかしアルファベット以外のキャラクタのビット5を操作しないように，事前にチェックしなければなりません．

6.4.3 データ変換(2)

2進データをアスキー・コードで表現した2進数，10進数，16進数に変換するプログラムを考えます．これらの変換は，主に出力ルーチンにおいて必要となります．

■ BIN TO ASC(BIN)

リスト14は，BA00H番地から10バイトのメモリの内容を2進数で表示するプログラムです．BIASBXでは，Cレジスタの内容を1ビットずつローテイトしながらCYフラグに取り出して，1であれば31Hに，0であれば30Hに変換してCRTに出力しています．

リスト14

```
                    ;      BIN TO ASC(BIN)
                    ;      3/23    BY M.F
                    ;
3E0D                PUTCHR:EQU  3E0DH      CRT出力ルーチンのエントリ・アドレス
BA00                DATA:  EQU  0BA00H     BA00Hからのメモリの内容を変換する
                    ;
                           ORG  0B900H
B900                START:
B900 2100BA                LD   HL,DATA   ┐
B903 060A                  LD   B,10      │
B905 CD21B9         L0:    CALL CRLF      │ BA00HからBA09Hまでのメモリ内容を,2進形式で出力する
B908 4E                    LD   C,(HL)    ┝ HLがポインタ，Bがカウンタとなっている
B909 23                    INC  HL        │ Cにデータを取り出してBIASBXを呼ぶ
B90A CD10B9                CALL BIASBX    │
B90D 10F6                  DJNZ L0        ┘
                    ;
B90F FF             EXIT:  RST  38H        モニタへ戻る
                    ;
B910 C5             BIASBX:PUSH BC         Bのカウンタを保存する
B911 0608                  LD   B,8        1バイトは8ビットであるからカウンタとして8を設定する
B913 3E30           BAB1:  LD   A,'0'      とりあえず30Hを設定しておく
B915 CB01                  RLC  C          CYフラグにビットデータを取り出す
B917 3001                  JR   NC,BAB2   ┐ CYフラグがセットされれば'1'(31H)を出力し，そうでなけれ
B919 3C                    INC  A         ┝ ば'0'(30H)を出力する
B91A CD0D3E         BAB2:  CALL PUTCHR    ┘
B91D 10F4                  DJNZ BAB1       8ビット出力すればループを抜ける
B91F C1                    POP  BC         保存したカウンタを取り出す
B920 C9                    RET
                    ;
B921 3E0D           CRLF:  LD   A,0DH     ┐
B923 CD0D3E                CALL PUTCHR    │
B926 3E0A                  LD   A,0AH     ┝ CRとLFを出力して改行する
B928 CD0D3E                CALL PUTCHR    │
B92B C9                    RET            ┘
                    ;
B92C                       END
```

■ BIN　TO　ASC(HEX)

リスト15は，BA00H番地からアドレスとその内容を16進数に変換して出力するプログラムです．

4ビットのデータが，アスキー・コードの1バイトになりますから，1バイトのデータから4ビットずつ取り出して変換を行っています．データが0000～1001の場合には30Hから39Hに対応しますから，30Hを足すことによって変換できますが，1010～1111の場合には41H～46Hに対応しますから，37Hを足さなければなりません．この例では，30Hは共通に足し1010～1111の場合にはさらに7を足すようにしています．

アドレスは2バイトで表しますから4バイトのアスキー・コードに，データは1バイトですから2バイトのアスキー・コードに変換されます．

このプログラムを応用することによってモニタのDコマンドと同じ働きをさせることができます．

リスト15

```
                    ;        BIN TO ASC(HEX)
                    ;        1/3     BY M.F
                    ;
3583                GETCHR:EQU  3583H     キー入力ルーチンのエントリ・アドレス
3E0D                PUTCHR:EQU  3E0DH     CRT出力ルーチンのエントリ・アドレス
BA00                DATA:  EQU  0BA00H    BA00Hからのメモリの内容を変換する
                    ;
                           ORG  0B900H
B900                START:
B900 2100BA                LD   HL,DATA
B903 0610                  LD   B,16
B905 CD38B9         L0:    CALL CRLF
B908 CD18B9                CALL ADHEX      BA00H~BA0FHのアドレスとその内容を16進数で出力する
B90B 3E20                  LD   A,' '      HLがポインタ，Bがカウンタになっている
B90D CD0D3E                CALL PUTCHR     ADHEXでアドレスを出力する．スペースをあけた後，BIASHX
B910 4E                    LD   C,(HL)     で内容を出力する
B911 23                    INC  HL
B912 CD21B9                CALL BIASHX
B915 10EE                  DJNZ L0
                    ;
B917 FF             EXIT:  RST  38H       モニタへ戻る
                    ;
B918 4C             ADHEX: LD   C,H
B919 CD21B9                CALL BIASHX
B91C 4D                    LD   C,L        HLの内容を変換して出力する
B91D CD21B9                CALL BIASHX
B920 C9                    RET
                    ;
B921                BIASHX:
B921 79                    LD   A,C
B922 0F                    RRCA
B923 0F                    RRCA
B924 0F                    RRCA            上位4ビットを変換して出力する
B925 0F                    RRCA
B926 CD2AB9                CALL BAH1
B929 79                    LD   A,C       下位4ビットを変換して出力する
B92A E60F           BAH1:  AND  0FH       下位4ビットのみ取り出す
B92C FE0A                  CP   10
B92E 3802                  JR   C,BAH2
B930 C607                  ADD  A,7        0~9であれば30Hを加えて出力する
B932 C630           BAH2:  ADD  A,30H      0AH~0FHであれば37Hを加えて出力する
B934 CD0D3E                CALL PUTCHR
B937 C9                    RET
                    ;
B938 3E0D           CRLF:  LD   A,0DH
B93A CD0D3E                CALL PUTCHR
B93D 3E0A                  LD   A,0AH      改行する
B93F CD0D3E                CALL PUTCHR
B942 C9                    RET
                    ;
B943                       END
```

■ **BIN TO ASC(DEC)**

次に，10進数への変換を考えます．アスキー・コードから2進データへの変換がそうであったように，2のべき乗を基数としない数への変換は少し手間がかかります．

リスト16とリスト17は，BA00H番地から16バイトのデータを10進数に変換して出力するプログ

ラムですが，変換の方法が異なります．

通常私たちが手計算で2進→10進変換を行うときには，図6.14のように各ビットの重みを10進数で考えて，それらの和をとって変換をします．この方法と全く同じ方法で変換を行うには，2の各べき乗に対するBCDの数値データとして用意しておいて，それをビットが1ならば足していき，最後にその結果をアスキー・コードに変換することになります．

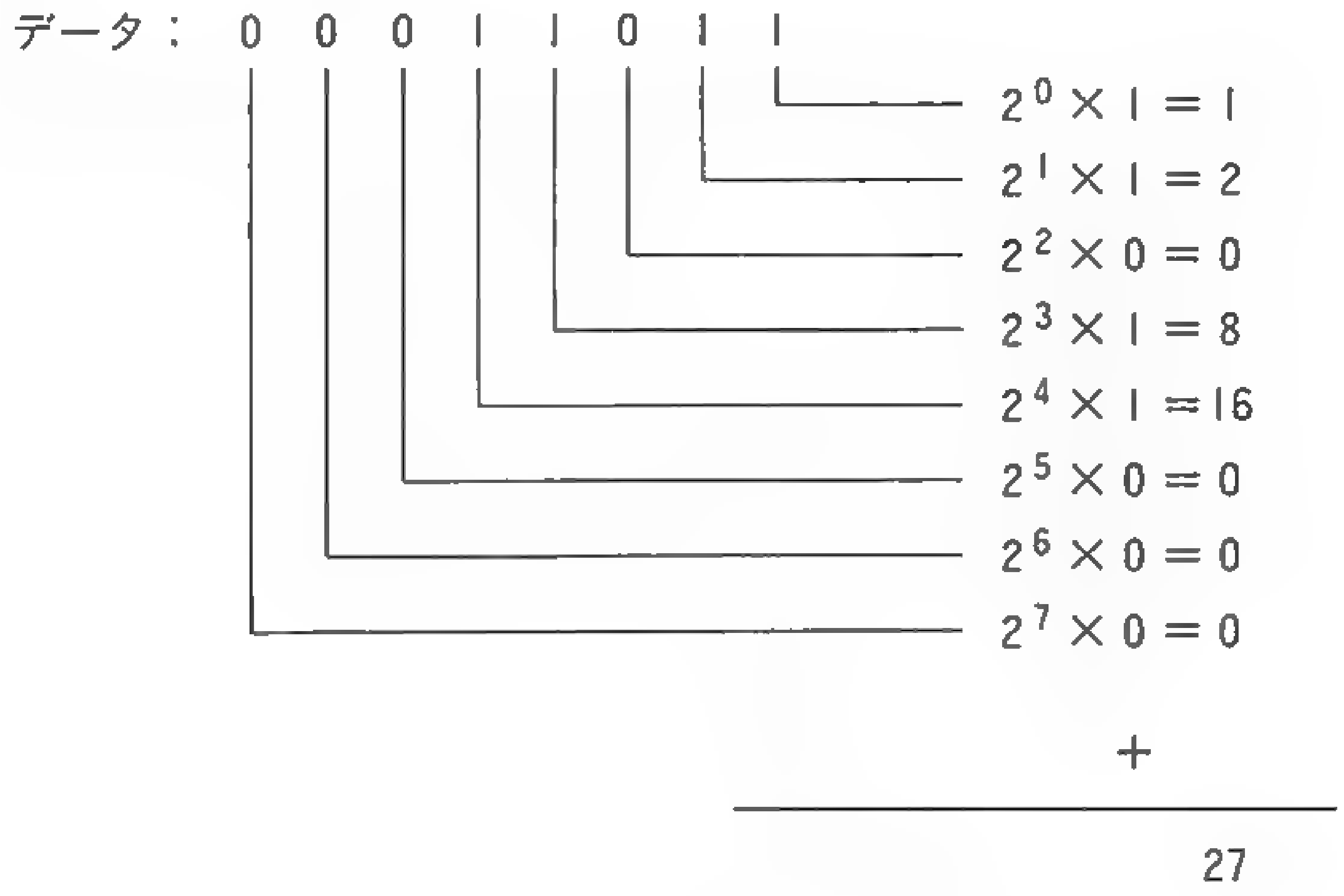


図6.14　2進→10進変換

リスト16は，基本的にはこれと同じことを行っているのですが，BCD演算を効果的に利用しているため，データを用意する必要がありません．このリスト中では，BIASDXで，Cレジスタの内容を10進変換して，DEレジスタ・ペアにBCD表現で格納しています．

B91CH番地からB923H番地までは，DEレジスタ・ペアのBCD表現のデータを2倍する部分です．DAAは，BCD演算には欠かせない便利な命令で，必要があれば，BCD桁に対して±6の補正(いわゆる10進補正)を行います．この命令に関しては後述することにします．

BIASDXの動作は，Cレジスタの内容をある1ビットだけセットされているとして考えてみるとよいでしょう．例えば，ビット7のみがセットされている80Hというデータであったとしますと，DEレジスタ・ペアには0128Hというデータが生成されます．これは，BCD表現ですから，128という10進数を表すことになります．128という数はもうお気付きかと思いますが，2^7のことです．他のビットにおいても同様に，そのビットに対応したBCDの数が生成されます．ここでは，これらのデータ生成と加算を同時に行って変換を行っています．

BCDASCは，BCDデータをアスキー・コードに変換して出力するルーチンです．BCDの1桁は0000～1001ですから，実はリスト15のBIASHXのルーチンと同じでもかまわないのですが，少し変えてあります．

リスト17は，これと全く異なったアルゴリズムで変換を行っています．これは，1バイトのデータは10進数に変換するとたかだか3桁であることを考えて，各桁を除算を行って得ています．つまり64H(100)で割った商が3桁目であり，その余りを0AH(10)で割った商が2桁目，その余りは1桁目となります．

この例では，除算を，減算を繰り返してその回数を数えて行っています．

BAD3のADD A, Bは，引き過ぎたデータを再び加えて正しいデータに直すための命令です．

リスト16

```
                    ;         BIN TO ASC(DEC) 1
                    ;         1/3       BY M.F
                    ;
3583                GETCHR:EQU  3583H     キー入力ルーチンのエントリ・アドレス
3E0D                PUTCHR:EQU  3E0DH     CRT出力ルーチンのエントリ・アドレス
BA00                DATA:  EQU  0BA00H    BA00Hからのメモリの内容を変換する
                    ;
                           ORG  0B900H
B900                START:
B900 2100BA                LD   HL,DATA
B903 0610                  LD   B,16
B905 CD28B9         L0:    CALL CRLF
B908 4E                    LD   C,(HL)     BA00HからBA0FHまでのメモリ内容を10進数で出力する
B909 23                    INC  HL         HLがポインタ，Bがカウンタになっている．BIASDXでBCD
B90A CD13B9                CALL BIASDX     HLに変換し，ADHEX1でそれをアスキー・コードに変換して
B90D CD33B9                CALL ADHEX1     出力する
B910 10F3                  DJNZ L0
                    ;
B912 FF             EXIT:  RST  38H       モニタへ戻る
                    ;
B913                BIASDX:
B913 C5                    PUSH BC
B914 0608                  LD   B,08H     2⁷~2⁰の8桁のデータを変換する
B916 110000                LD   DE,0      DEに結果が入いるのでクリアしておく
B919 AF                    XOR  A         CYフラグをクリアする
B91A CB21           BASD1: SLA  C
B91C 7B                    LD   A,E
B91D 8F                    ADC  A,A
B91E 27                    DAA             ビットデータをCYフラグに出す
B91F 5F                    LD   E,A        DEの内容をBCD演算で2倍すると共にCからのデータをCYフ
B920 7A                    LD   A,D        ラグが受け取る
B921 8F                    ADC  A,A        DEにBCDの型で変換結果が入る
B922 27                    DAA
B923 57                    LD   D,A
B924 10F4                  DJNZ BASD1
B926 C1                    POP  BC
B927 C9                    RET
                    ;
B928 3E0D           CRLF:  LD   A,0DH
B92A CD0D3E                CALL PUTCHR
B92D 3E0A                  LD   A,0AH      改行する
B92F CD0D3E                CALL PUTCHR
B932 C9                    RET
                    ;
B933 C5             ADHEX1:PUSH BC
B934 4A                    LD   C,D
B935 CD3EB9                CALL BCDASC
B938 4B                    LD   C,E        DEの内容をアスキー・コードにして出力する
B939 CD3EB9                CALL BCDASC     DEには，BCDで4桁のデータが入っている
B93C C1                    POP  BC
B93D C9                    RET
```

```
                 ;
B93E 79          BCDASC:LD    A,C
B93F 0F                 RRCA
B940 0F                 RRCA          上位4ビットを変換して出力する
B941 0F                 RRCA
B942 0F                 RRCA
B943 CD47B9             CALL  BCD1
B946 79                 LD    A,C     下位4ビットを変換して出力する
B947 E60F        BCD1:  AND   0FH     下位4ビットのみ取り出す
B949 C630               ADD   A,30H   データは0～9であるから30Hを加えればアスキー・コードとな
B94B CD0D3E             CALL  PUTCHR  り，それを出力する
B94E C9                 RET
                 ;
B94F                    END
```

リスト17

```
                 ;      BIN TO ASC(DEC) 2
                 ;      3/18    BY M.F
                 ;
3583.            GETCHR:EQU   3583H    キー入力ルーチンのエントリ・アドレス
3E0D             PUTCHR:EQU   3E0DH    CRT出力ルーチンのエントリ・アドレス
BA00             DATA:  EQU   0BA00H   BA00Hからのメモリの内容を変換する
                 ;
                        ORG   0B900H
B900             START:
B900 2100BA             LD    HL,DATA
B903 0610               LD    B,16
B905 CD33B9      L0:    CALL  CRLF     BA00HからBA0FHまでのメモリ内容を10進数で出力する
B908 4E                 LD    C,(HL)   HLがポインタ，Bがカウンタになっている
B909 23                 INC   HL       BIASD2で変換してCRTに出力する
B90A CD10B9             CALL  BIASD2
B90D 10F6               DJNZ  L0
                 ;
B90F FF          EXIT:  RST   38H      モニタへ戻る
                 ;
B910             BIASD2:
B910 C5                 PUSH  BC
B911 0664               LD    B,100    100の位(10進)を求め，CRTに出力する    Cの内容を3桁の
B913 CD21B9             CALL  BAD1                                          アスキー・コード
B916 060A               LD    B,10     10の位(10進)を求め，CRTに出力する     に変換して出力す
B918 CD21B9             CALL  BAD1                                          る
B91B 79                 LD    A,C      1の位(10進)を求め，CRTに出力する
B91C CD2DB9             CALL  BAD4
B91F C1                 POP   BC
B920 C9                 RET
                 ;
B921 79          BAD1:  LD    A,C
B922 1E00               LD    E,0
B924 90          BAD2:  SUB   B
B925 3803               JR    C,BAD3   各位の値を求めるため，その位の重み（100，10，1）で除算し
B927 1C                 INC   E        て商を求める
B928 18FA               JR    BAD2     除算は，減算の回数を数えて行う
                 ;                     BAD3のADD　A，Bは引きすぎた数をもとに戻すために必要
B92A 80          BAD3:  ADD   A,B      である．
B92B 4F                 LD    C,A
B92C 7B                 LD    A,E
B92D C630        BAD4:  ADD   A,30H
B92F CD0D3E             CALL  PUTCHR   アスキー・コードにして出力する
B932 C9                 RET
```

```
                   ;
B933 3E0D          CRLF:  LD    A,0DH      ┐
B935 CD0D3E               CALL  PUTCHR     │
B938 3E0A                 LD    A,0AH      ├CR, LFを出力して改行する
B93A CD0D3E               CALL  PUTCHR     │
B93D C9                   RET              ┘
                   ;
B93E                      END
```

■ DAA命令

最後になりましたが，DAA命令とBCD演算についてまとめておきます．

DAA命令は２進数の加算，減算の直後に使用し，Aレジスタの内容に対して正しいBCD表現になるように補正を行うものです．

例えば，加算の場合BCDの１桁(４ビット単位)からの桁上がりがあるときとデータが1001を超えるときにその桁に対して＋６補正を行うわけですが，DAA命令を使うと１命令でこれらのことを行ってくれます．

DAA命令動作中に，CYフラグ，Hフラグ，Nフラグが参照されますが，私たちは結果として，CYフラグとAレジスタの内容についてのみ考えれば充分です．

BCD演算を用いると，10進数の加算，減算を簡単に行うことができます．リスト18は，８桁のBCD加／減算プログラムです．このように，ADC／SBC命令で，CYフラグを含んだ演算を行うことによって何桁の演算でも作ることができます．ただし，この例では最上位桁の桁上がり，桁借りは考えていません．

リスト18

```
                   ;         BCD ADD 1 (SUB)
                   ;         3/31     BY M.F
                   ;
0004               DSIZE: EQU   4                     4バイト（BCD8桁）のデータを扱う
                   ;
                          ORG   0B900H
B900               START:
B900 111DB9               LD    DE,SDATA+DSIZE-1      DEに足される数の最下位のデータがあるアドレスを
                                                      セットする
B903 2121B9               LD    HL,DDATA+DSIZE-1      HLに足す数の最下位のデータがあるアドレスをセッ
                                                      トする
B906 DD2125B9             LD    IX,ODATA+DSIZE-1      IXに答が入るエリアの最終アドレスをセットする
B90A 0604                 LD    B,DSIZE               Bにデータのサイズをセットする（バイト単位）
B90C AF                   XOR   A                     CYフラグをクリアする
B90D               L0:
B90D 1A                   LD    A,(DE)                足されるの数のデータを取り出す
                   ;
B90E 8E                   ADC   A,(HL)               }CYフラグと共に加算（減算）する
                   ;     /SBC      A,(HL)/
                   ;
B90F 27                   DAA                         BCD演算における桁上り（借り）があればそれを補
B910 DD7700               LD    (IX),A                正する答えを格納する
                   ;
B913 1B                   DEC   DE                    ┐
B914 2B                   DEC   HL                    │
B915 DD2B                 DEC   IX                    ├次の桁のデータを取り出すためポインタをデクリメ
B917 10F4                 DJNZ  L0                    ┘ントする
```

```
                      ;
B919 FF               EXIT:  RST   38H                モニタへ戻る
                      ;
B91A 12345678 SDATA:  DB    12H,34H,56H,78H    足される数のデータ ┐ BCD最上位からの桁上げ(借り)
                      ;                                              │ は，考えていないため，注意が
                                                                     │ 必要である．減算では，引かれ
B91E 11998844 DDATA:  DB    11H,99H,88H,44H    足す数のデータ     ┘ る数が引く数より大きくなけれ
                                                                       ばならない
                      ;
B922                  ODATA: DS    DSIZE              答えが格納されるエリアを確保する
                      ;
B926                         END
```

6.5 I/Oポートを使う

μPD780は，メインのアドレス空間とは別に，I/Oのアドレス空間として256バイトをもっています．そしてこのアドレス空間を，IN命令，OUT命令でアクセスすることは前述した通りです(第5章参照)．

ここではまず入力としてキースキャン，出力としてプリンタ出力を取り上げ，次にテキスト・エリアを読んでみることにします．

6.5.1　キースキャン

キースキャンとは，キーボードのどのキーが押されるかを調べることです．一般にキーボードからのデータを受け取るには内部ルーチンの3583Hや35CEHなどを呼べばよいのですが，特殊な用途，つまりゲームなどで特定のキー入力を高速に調べる必要がある場合はキースキャンは有効です．

PC-8801のキーボードは本体から分離していますが，このキーボードの中は，ICが1つと，あとはスイッチが並んでいるだけの簡単なものです．そして私たちが扱うキーボードからのデータは，I/Oポートの00H～0BHに現れます．キーとポートとデータの関係を図6.15に示します．

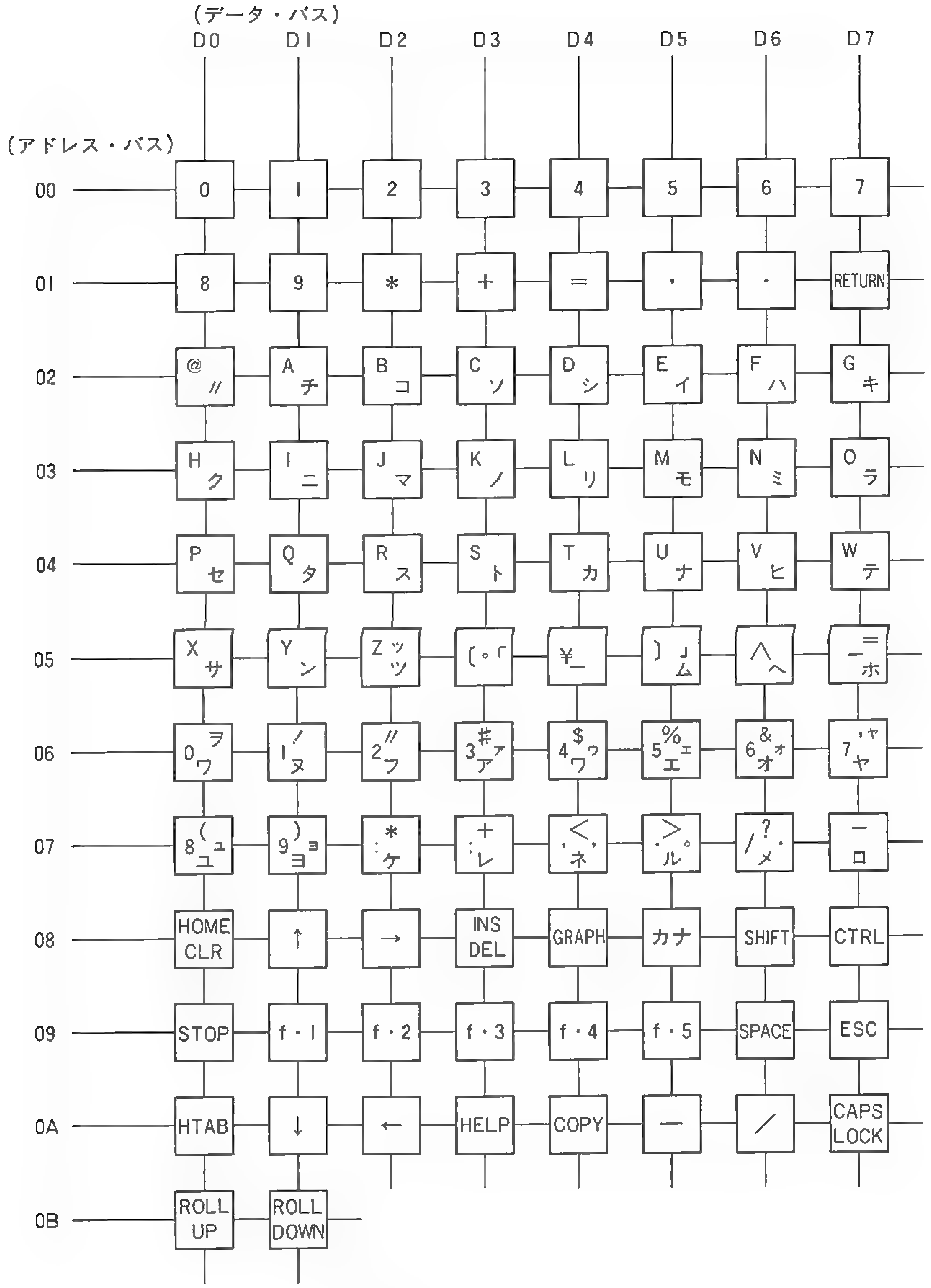
例えばこの図で0のキーは，ポート00Hの入力データの0ビットに対応していることがわかります．そして，このビットが1のときは押されていない状態，0のときは押されている状態を示します．つまり，

```
    IN      A,(00)
```

の後のAレジスタの内容がFEHであれば，0のキーは押されていたことになります．

ここで気付かれたと思いますが，このデータは0のアスキー・コードの30Hではないということです．ですから，もしアスキー・コードでデータを得たい場合は変換しなければなりません．

ではここで，キースキャンの例としてカーソル・コントロールキーの入力プログラムをリスト19とリスト20に示します．



(アドレス・バス)	D0	D1	D2	D3	D4	D5	D6	D7
00	0	1	2	3	4	5	6	7
01	8	9	*	+	=	,	.	RETURN
02	@ ″	A チ	B コ	C ソ	D シ	E イ	F ハ	G キ
03	H ク	I ニ	J マ	K ノ	L リ	M モ	N ミ	O ラ
04	P セ	Q タ	R ス	S ト	T カ	U ナ	V ヒ	W テ
05	X サ	Y ン	Z ッ ツ	[° 「	¥ _	] 」 ム	^ ヘ	= — ホ
06	0 ヲ ワ	1 ! ヌ	2 ″ フ	3 # ァ ア	4 $ ゥ ウ	5 % ェ エ	6 & ォ オ	7 ' ャ ヤ
07	8 (ュ ユ	9) ョ ヨ	: * ケ	; + レ	, < 、 ネ	. > 。 ル	/ ? ・ メ	— ロ
08	HOME CLR	↑	→	INS DEL	GRAPH	カナ	SHIFT	CTRL
09	STOP	f・1	f・2	f・3	f・4	f・5	SPACE	ESC
0A	HTAB	↓	←	HELP	COPY	—	／	CAPS LOCK
0B	ROLL UP	ROLL DOWN						

図6.15 キーマトリックス

リスト19

```
                   ;        CURSOR SCAN 1
                   ;        3/16    BY M.F
                   ;
0008               PORT08:EQU   08H       }キー入力ポート0～0CHのうちカーソル・コントロールキー,
0009               PORT09:EQU   09H       }STOPキーに関係するポート
000A               PORT0A:EQU   0AH
3E0D               PUTCHR:EQU   3E0DH     CRT出力ルーチンのエントリ・アドレス
                   ;
                           ORG   0B900H
```

```
B900          START:
B900 DB09             IN    A,(PORT09)   STOPキーが押されていれば，EXITからモニタへ戻る．
B902 FEFE             CP    0FEH         押されていなければSTART 1へ飛ぶ
B904 2001             JR    NZ,START1
              ;
B906 FF       EXIT:   RST   38H
              ;
B907 DB08     START1:IN     A,(PORT08)   ポート08のデータを取り込みBへロードする
B909 47               LD    B,A
B90A DB0A             IN    A,(PORT0A)   ポート0AHのデータを取り込みCへロードする
B90C 4F               LD    C,A
B90D CB48             BIT   1,B          ↑キーが押されていればUP 1を呼ぶ
B90F CC23B9           CALL  Z,UP1
B912 CB50             BIT   2,B          →キーが押されていればRIGHT 1を呼ぶ
B914 CC2DB9           CALL  Z,RIGHT1
B917 CB49             BIT   1,C          ↓キーが押されていればDOWN 1を呼ぶ
B919 CC28B9           CALL  Z,DOWN1
B91C CB51             BIT   2,C          ←キーが押されていればLEFT 1を呼ぶ
B91E CC32B9           CALL  Z,LEFT1
B921 18DD             JR    START        STARTへ飛びループになる
              ;
B923 3E55     UP1:    LD    A,'U'        'U'を出力する
B925 C30D3E           JP    PUTCHR
B928 3E44     DOWN1:  LD    A,'D'        'D'を出力する   PUTCHRの終わりのRET命令でリターン
B92A C30D3E           JP    PUTCHR                       する
B92D 3E52     RIGHT1:LD     A,'R'        'R'を出力する
B92F C30D3E           JP    PUTCHR
B932 3E4C     LEFT1:  LD    A,'L'        'L'を出力する
B934 C30D3E           JP    PUTCHR
              ;
B937                  END
```

リスト20

```
              ;       CURSOR SCAN 2
              ;       3/16     BY M.F
              ;
0008          PORT08:EQU  08H          キー入力ポート0～0BHのうちカーソル・コントロールキー，
0009          PORT09:EQU  09H          STOPキーに関するポート
000A          PORT0A:EQU  0AH
3E0D          PUTCHR:EQU  3E0DH        CRT出力ルーチンのエントリ・アドレス
              ;
                      ORG  0B900H
B900          START:
B900 CD36B9           CALL INPUT       STOPキーが押されていれば，EXITへ飛ぶ．カーソルコン
B903 3848             JR   C,EXIT      トロールキーが押されていなければ，START へ飛びループ
B905 28F9             JR   Z,START     になる．どのカーソルが押されているかのデータがAの下位4
                                       ビットにある
              ;
B907 0F               RRCA
B908 3003             JR   NC,L1       ↓キーが押されていればDOWN 1を呼ぶ
B90A CD21B9           CALL DOWN1
B90D 0F       L1:     RRCA
B90E 3003             JR   NC,L2       ←キーが押されていればLEFT 1を呼ぶ
B910 CD25B9           CALL LEFT1
B913 0F       L2:     RRCA
B914 3003             JR   NC,L3       ↑キーが押されていればUP 1を呼ぶ
B916 CD29B9           CALL UP1
B919 0F       L3:     RRCA
B91A 3003             JR   NC,L4       →キーが押されていればRIGHT 1を呼ぶ
B91C CD2DB9           CALL RIGHT1
B91F 18DF     L4:     JR   START       STARTへ飛びループになる
```

```
                        ;
B921 0E44       DOWN1:  LD     C,'D'          }'D' を出力する
B923 180A               JR     OUTC1
B925 0E4C       LEFT1:  LD     C,'L'          }'L' を出力する
B927 1806               JR     OUTC1
B929 0E55       UP1:    LD     C,'U'          }'U' を出力する
B92B 1802               JR     OUTC1
B92D 0E52       RIGHT1:LD      C,'R'          }'R' を出力する
                ;
B92F 47         OUTC1:  LD     B,A
B930 79                 LD     A,C
B931 CD0D3E             CALL   PUTCHR         Cのデータを出力するルーチン．Aは保存される
B934 78                 LD     A,B
B935 C9                 RET
                ;
B936            INPUT:
B936 DB09               IN     A,(PORT09)
B938 E601               AND    01H            STOPキーが押されていればCY=1にしてリターンする．
B93A 2002               JR     NZ,INP1        そうでなければINP1へ飛ぶ
B93C 37                 SCF
B93D C9                 RET
                ;
B93E DB08       INP1:   IN     A,(PORT08)
B940 2F                 CPL
B941 E606               AND    06H
B943 07                 RLCA                  Aの下位4ビットが4つのカーソル・コントロールキーに対応
B944 47                 LD     B,A            するCPLで反転しているから，ビットがセットされていれば
B945 DB0A               IN     A,(PORT0A)     押されていることになる
B947 2F                 CPL                   CY=0にしてリターンする
B948 E606               AND    06H
B94A 0F                 RRCA
B94B B0                 OR     B
B94C C9                 RET
                ;
B94D FF         EXIT:   RST    38H            モニタへ戻る
                ;
B94E                    END
```

リスト19の方はBIT命令で入力データを調べて直接それぞれの処理ルーチンへ移っています．

リスト20は，INPUTルーチンでAレジスタの下位4ビットにカーソル入力のデータを作成し（図6.16），その後データをCYフラグに取り出しながら調べていって，それぞれの処理を行っています．また，このようにキーが押されているときにビットがリセットされるような場合，CPL命令などでデータを反転すると調べやすくなります．

	MSB								LSB
PORT08Hのデータ							→	↑	
Aレジスタの内容	0	0	0	0	→	↑	↑	←	↓
PORT0AHのデータ							←	↓	

図6.16　INPUTルーチンの動作

6.5.2　プリンタ出力

プリンタに1文字出力するには内部ルーチン3ED4Hを呼べば済むことですが，実際に，ポートにデータを出力してプリンタを動かしてみましょう(以後，PC-8821，PC-8822，プリンタユーザーズ·マニュアルをもとに記述していきますが，本質的には，PC-8023でもエプソン社のMPシリーズでも同じように扱うことができます).

プリンタに送るデータは，普通テキストVRAMに出力するものと同じキャラクタ·コードです.このデータの送り方には，一度に8ビットのデータを送るパラレル方式と，1ビットずつ送るシリアル方式がありますが，PC-8801ではパラレル方式を採用しています.よく聞くセントロニクス規格というのは，このパラレル方式に関する規格の1つです．8ビットをパラレルで送るというのは，OUT命令で一度に扱うデータは1バイト，つまり8ビットなのですから，それ程特別なことではありません.

このキャラクタ·コードの8ビット·データは，ポートの10Hに出力します．ここに出力したデータは，プリンタケーブルを通って，プリンタ・ユーザーズ・マニュアルのコネクタ接続図におけるDATA1～DATA8に現れます.

PIN No.	信号名	PIN No.	信号名
1	$\overline{\text{DATA·STB}}$	19	TWISTED PAIR GND (PIN 1)
2	DATA 1	20	(2)
3	2	21	(3)
4	3	22	(4)
5	4	23	(5)
6	5	24	(6)
7	6	25	(7)
8	7	26	(8)
9	DATA 8	27	(9)
10	$\overline{\text{ACK}}$	28	(10)
11	BUSY	29	(11)
12	PE	30	TWISTED PAIR GND (31)
13	SELECT	31	$\overline{\text{INPUT-PRIME}}$
14	0V	32	$\overline{\text{FAULT}}$
15	NC	33	0V
16	0V	34	NC
17	CHASSIS GND	35	NC
18	+5V DC	36	BUSY

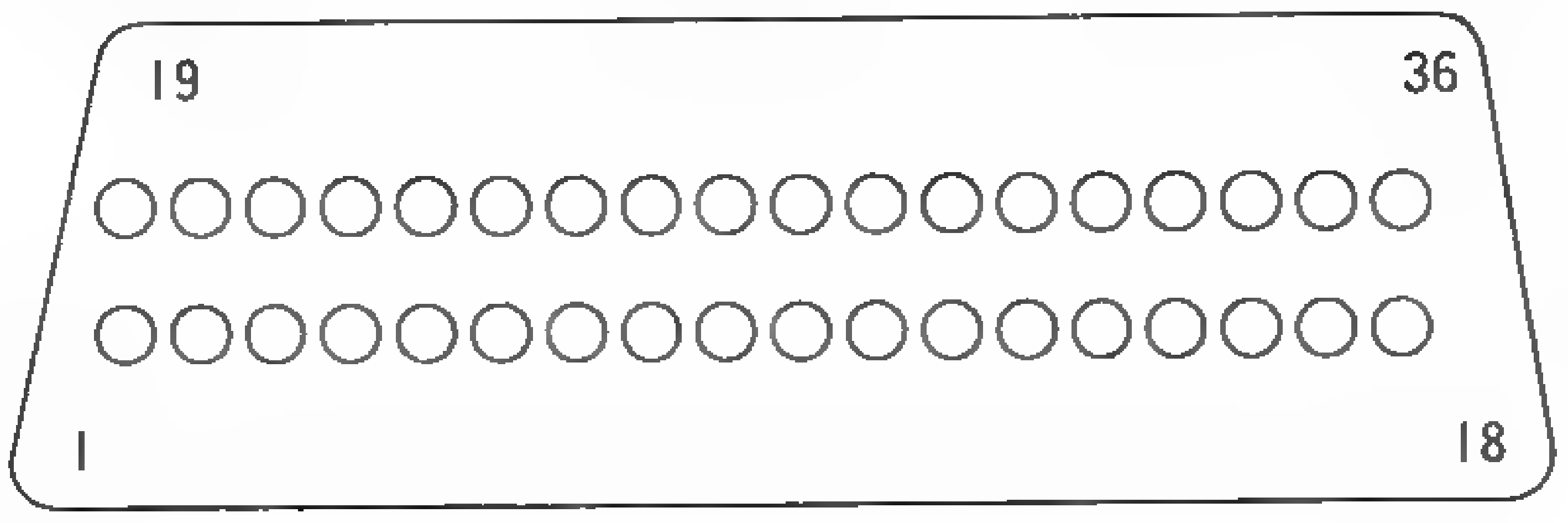


図6.17　コネクタ接続図

さて，この図をみると，他にもいろいろな信号があることがわかりますが，そのうちここで使用するものはDATA・STB(データ・ストローブ)と，BUSY(ビジイ)です．

まずBUSY信号ですが，これは，プリンタが今データを受信不可能であることをPC-8801に伝えるための信号です．つまり，この信号がアクティブ(Highレベル)にあるときは，プリンタにデータを送ってはいけないことを示しています．

データ・ストローブ信号は，プリンタに対してデータが用意できたことを示す信号です．プリンタは，この信号(パルス)を捕えてデータを内部バッファに取り込みます．

PC-8801では，BUSY信号は入力ポートの40Hのビット0に，ストローブ信号は出力ポートの40Hのビット0に対応します．ここで注意しなければならないのは，この40Hのポートが，入力と出力で全く意味が異なるということです．そこでN_{88}-BASICでは，40Hに出力したデータと同じものをE6C1H番地に格納しています．また，プリンタに関するものはビット0だけですから，その他のビットに影響を与えてはいけません．

ストローブ信号を送るタイミングは，図6.18のように決まっています(PC-8821／22の場合)．つまり，データとしては最低3μsの間その内容を変更してはならず，ストローブ信号として，最低1μsの信号を図の位置に出力しなければなりません．

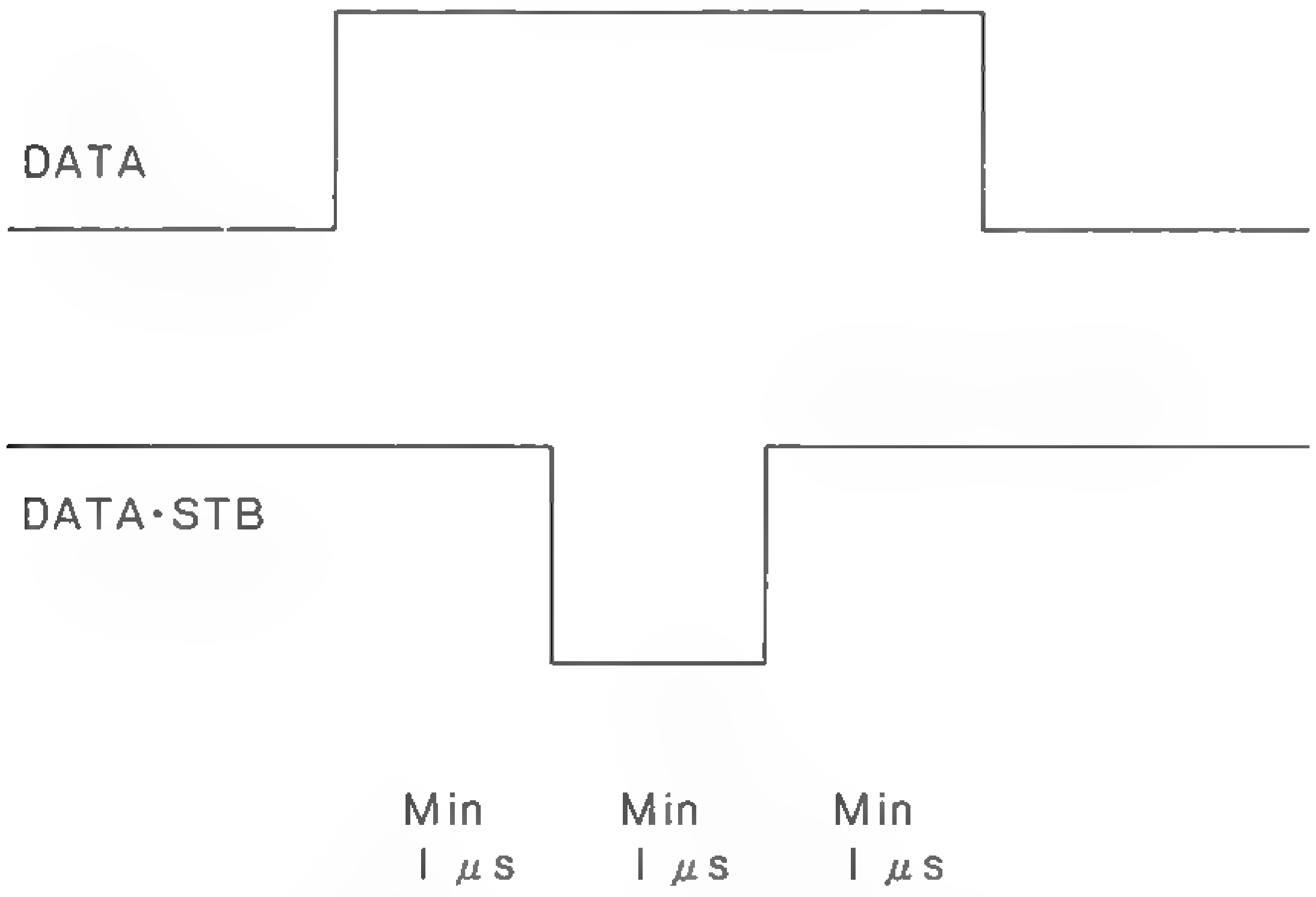


図6.18 データとストローブ信号

1μs(1マイクロ秒：10^{-6}秒)という時間はどれくらいなのかというと，これは，μPD780C-1がクロック4MHzで動作しているとき，最も速く処理を終わる命令(4ステートのもの：NOP, LD r, r′…etc)に要する時間であると覚えておけばよいでしょう．

ところで，これだけではまだプリンタは正常に動作してくれません．一般にプリンタが印字を始めるのはプリンタ内のバッファ(機種によって容量が異なる)がいっぱいになったときか，キャリッジリターン・コード：0DHが送られたとき(0AHのみでもよい場合がある)です．バッファは最低でも1行分はありますので，行の終わりに0DH, 0AHを送るようにします．

リスト21は，AからZまでの文字を10行印字するプログラムです．図6.19にOPRN1のフローを示します．

このプログラムだけなら割込みに関するDI，EI命令を省略してもかまいませんが，このようにI/Oを直接操作する場合には，割込みを禁止するほうが無難です．

リスト21

```
                  ;        TEST PRNT 1
                  ;        3/17    BY M.F
                  ;
0010              PORT10:EQU   10H         プリンタヘデータを送る出力ポート
0040              PORT40:EQU   40H         各種のコントロール信号のI／Oポート．ここではビット0のみ使う
E6C1              DPRT40:EQU   0E6C1H      出力ポート40Hに出力したデータが保存してあるアドレス
                  ;
                         ORG   0B900H
B900              START:
B900 060A                LD    B,10        10行出力するためのカウンタ値をセットする
B902 3E41         L1:    LD    A,'A'      ┐
                  ;                        │
B904 CD19B9       L0:    CALL  OPRN1       │'A'から'Z'までの26文字をプリンタに出力する
B907 3C                  INC   A           │
B908 FE5B                CP    'Z'+1       │
B90A 38F8                JR    C,L0        ┘
                  ;
B90C 3E0D                LD    A,0DH       ┐
B90E CD19B9              CALL  OPRN1       │CR，LFを出力して改行する（1行印字）
B911 3E0A                LD    A,0AH       │
B913 CD19B9              CALL  OPRN1       ┘
B916 10EA                DJNZ  L1          10行出力したらループを抜ける
                  ;
B918 FF           EXIT:  RST   38H         モニタへ戻る
                  ;
B919 F5           OPRN1: PUSH  AF          ┐カウンタとデータを保存する
B91A C5                  PUSH  BC          ┘
B91B 0E40                LD    C,PORT40
B91D ED40         OP1:   IN    B,(C)       ┐プリンタがBUSY（ビジイ）でなくなるまでループする
B91F CB40                BIT   0,B         │（READYになるまで待つ）
B921 20FA                JR    NZ,OP1      ┘
                  ;
B923 D310                OUT   (PORT10),A  データをプリンタに出力する
                  ;
B925 F3                  DI
B926 3AC1E6              LD    A,(DPRT40)  ┐
B929 E6FE                AND   0FEH        │
B92B ED79                OUT   (C),A       │プリンタヘストローブ信号を送る
                  ;                        │40Hに出力してあるデータのうち，ビット0だけを変化させる
B92D CBC7                SET   0,A         │
B92F ED79                OUT   (C),A       ┘
                  ;
B931 FB                  EI
B932 C1                  POP   BC          ┐
B933 F1                  POP   AF          │保存したデータを取り出す
B934 C9                  RET               ┘
                  ;
B935                     END
```

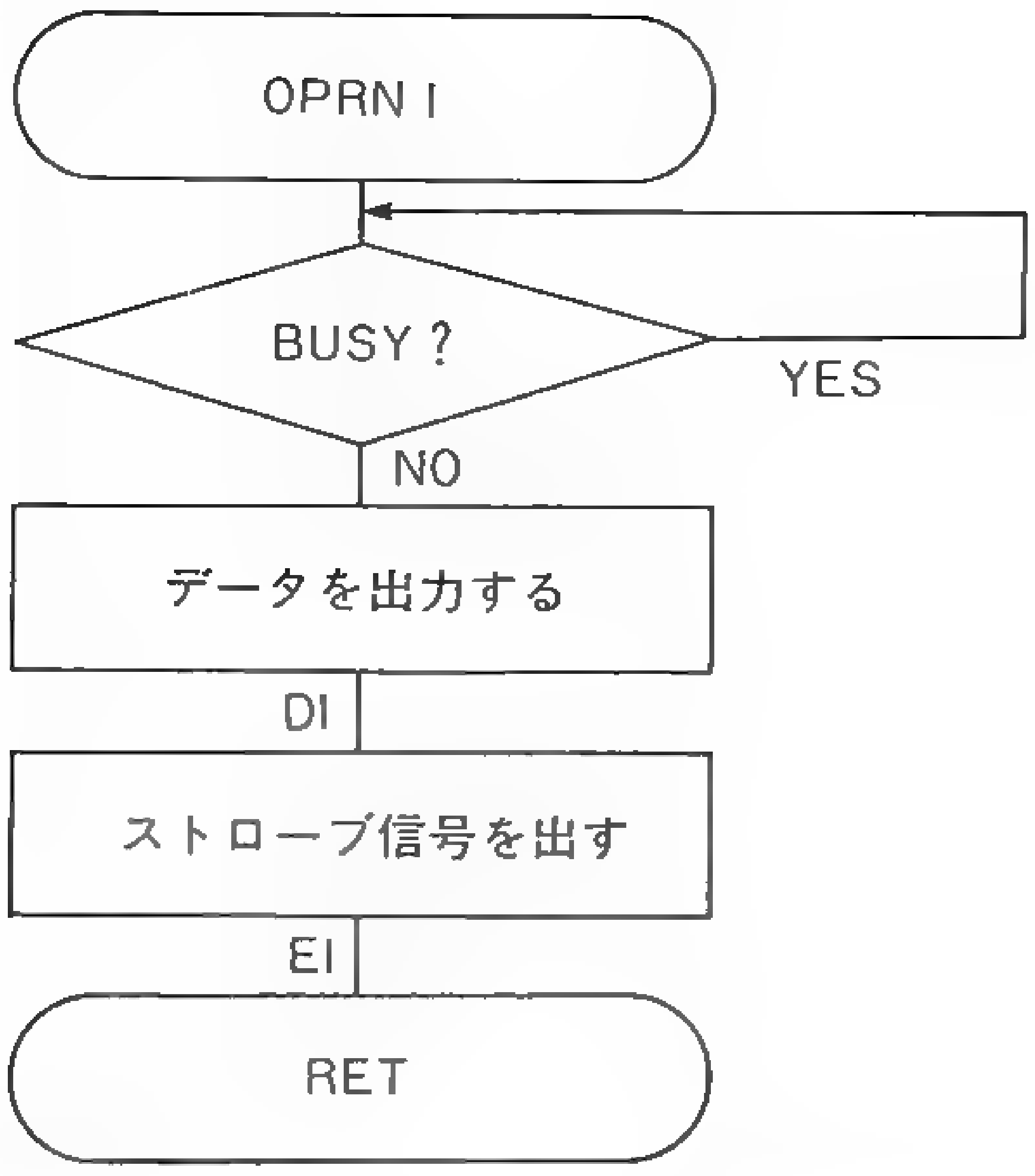


図6.19　プリンタ出力ルーチンの流れ

内部ルーチン3ED4H番地は，基本的にはOPRN1と変わりません．ただ，プリンタにデータを送る前に STOP の入力があったかどうかを間接的に調べ，入力があったならそのまま何もせずに戻ります．

6.5.3　テキストを読む

PC-8801は，メモリ空間を有効に使うために，0000H番地～7FFFH番地のアドレスではROMバンクとRAMバンクを切り換えて使用しています．ROMにはBASICインタプリタなどのプログラムが格納されており，RAMバンクにはN_{88}-BASIC動作時に私たちユーザが入力するBASICのテキストが，中間言語の形で格納されます．

N_{88}-BASIC動作時にはN_{88}-BASIC ROMがセレクトされており，RAMバンク(以後テキスト・エリア)はそのままでは読むことはできませんが，書き込むことは可能です．

テキスト・エリアを読むには二通りの方法があります．1つは，バンク切り換えを行ってRAMバンクをセレクトすることです．切り換え操作は簡単に行うことができますが，その場合には割込み等の管理をすべて自分で行わなければならないこともありますから注意してください．

もう1つは，PC-8801に用意されている特殊なハードウェアを利用する方法です．PC-8801には，BASIC ROMと同じアドレスにあるテキスト・エリアを読むためにテキスト・ウィンドウと呼ばれる領域があります．これは，8000H番地～83FFH番地の1Kバイトで，この領域からテキスト・エリアの0000H～7FFFHをのぞきみることができます．このエリアは32Kバイトありますから，1Kバイトのテキスト・ウィンドウから一度にすべてのテキスト・エリアの内容をみることはできません．そこで，上位8ビットのオフセットを与えてアドレスを設定します．このアドレスは，出力ポート70Hに設定します．例えば0を出力ポート70Hに出力した場合には，テキスト・エリアの0000H番地～03FFH番地の内容が8000H番地～83FFH番地のウィンドウに現われます．

テキスト・ウィンドウに現われている部分のテキスト・エリアの書き込みも，このウィンドウ

に対してデータを書くことにより行うことができます.

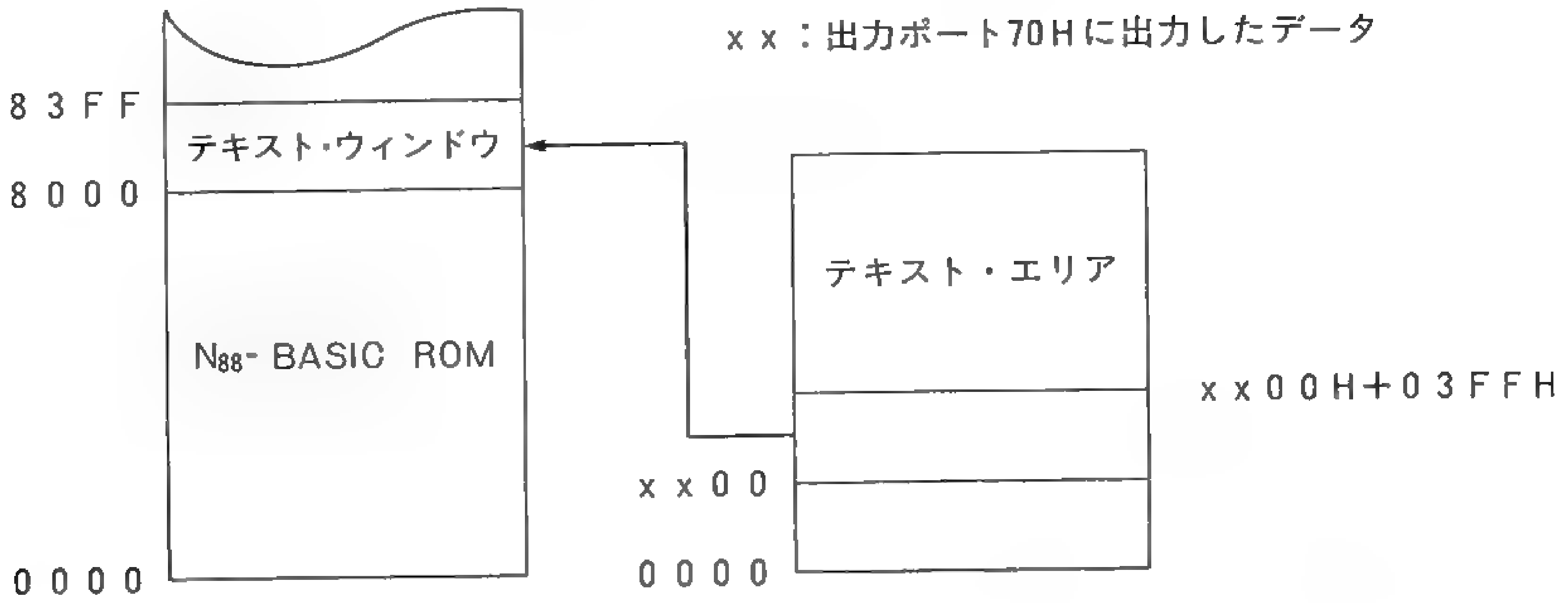

83FF
テキスト・ウィンドウ
8000
N88-BASIC ROM
0000
xx：出力ポート70Hに出力したデータ
テキスト・エリア
xx00H+03FFH
xx00
0000

図6.20　テキスト・ウィンドウとテキストRAM

テキストを読むにあたってまず知らなければならないのは，BASICのソース・プログラムがテキスト・エリア中にどのようなデータ構造で格納されているかということですが，詳しくは『Tech-know』にありますから，ここでは図6.21に示すだけにします.

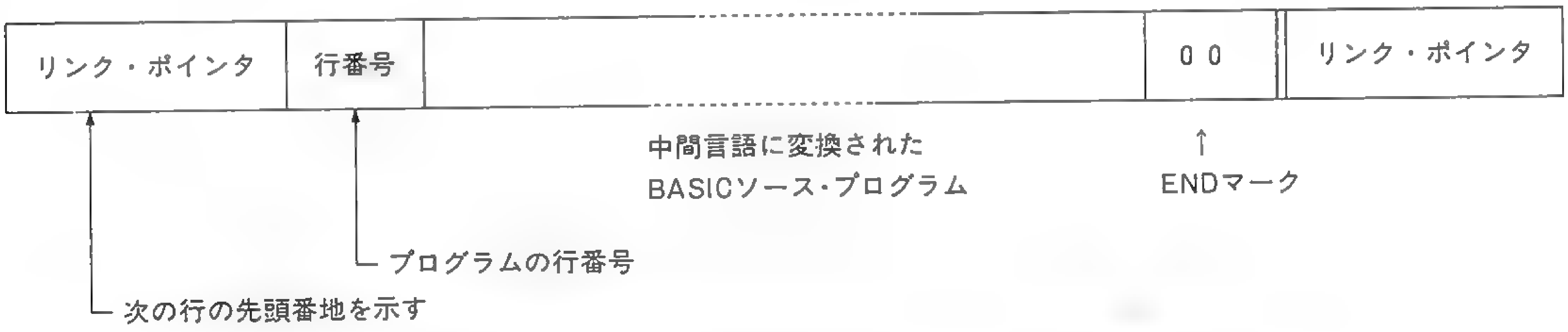

リンク・ポインタ
行番号
00
リンク・ポインタ
中間言語に変換された
BASICソース・プログラム
ENDマーク
プログラムの行番号
次の行の先頭番地を示す

図6.21　テキストRAM内のBASICテキストのデータ構造

リスト22

```
                  ;        READ TEXT RAM /REM 1
                  ;        3/18    BY M.F
                  ;
0070              OPRT70:EQU   70H
8000              SWINDW:EQU   8000H        テキスト・ウィンドウのスタート・アドレス
3E0D              PUTCHR:EQU   3E0DH        CRT出力ルーチンのエントリ・アドレス
E658              TXTSTA:EQU   0E658H       テキストスタート・アドレスの入っているワーク・エリア
EB18              TXTEND:EQU   0EB18H       テキストエンド・アドレスの入っているワーク・エリア
                  ;
                         ORG   0B900H
B900              START:
B900 2A58E6              LD    HL,(TXTSTA)  HLにテキストスタート・アドレスをセットする
B903 CD52B9       L0:    CALL  REAWIN       ウィンドウ・オフセットの設定とアドレス変換を行う
B906 CD58B9              CALL  GETDE        リンク・ポインタ（2バイト）を取り出す
B909 7B                  LD    A,E         ┐
B90A B2                  OR    D           │
B90B 2001                JR    NZ,L1       ├テキストの終わりであればモニタへ戻る
                  ;                        │
B90D FF           EXIT:  RST   38H         ┘
                  ;
B90E ED5398B9     L1:    LD    (LINKNO),DE  リンク・ポインタを保存する
                  ;
B912 CD58B9              CALL  GETDE       ┐
B915 ED539AB9            LD    (LINENO),DE ┘行番号を取り出し，保存する
```

```
                    ;
B919 7E                      LD    A,(HL)
B91A FE8F                    CP    8FH
B91C 2008                    JR    NZ,SREM1
B91E 23                      INC   HL
B91F 7E                      LD    A,(HL)
B920 FE20                    CP    ' '
B922 2812                    JR    Z,SREM2
B924 1826                    JR    NEXT
                    ;                               8FH，20Hと続けばREM文であるからSREM2へ飛ぶ
B926 FE3A           SREM1:   CP    ':'              3AH，8FH，0E9Hと続けばアポストロフィーによるRE
B928 2022                    JR    NZ,NEXT          M文であるからSREM2へ進む
B92A 23                      INC   HL               それ以外であれば，注釈行ではないとして次の行を調べる
B92B 7E                      LD    A,(HL)
B92C FE8F                    CP    8FH
B92E 201C                    JR    NZ,NEXT
B930 23                      INC   HL
B931 7E                      LD    A,(HL)
B932 FEE9                    CP    0E9H
B934 2016                    JR    NZ,NEXT
B936 23             SREM2:   INC   HL
                    ;
B937 E5                      PUSH  HL
B938 2A9AB9                  LD    HL,(LINENO)      行番号を10進5桁に変換して出力する
B93B CD5DB9                  CALL  BIASD2
B93E E1                      POP   HL
                    ;
B93F 7E             WR1:     LD    A,(HL)
B940 B7                      OR    A
B941 2806                    JR    Z,WR2
B943 CD0D3E                  CALL  PUTCHR           その行の内容を表示する
B946 23                      INC   HL
B947 18F6                    JR    WR1
                    ;
B949 CD8DB9         WR2:     CALL  CRLF
                    ;
B94C 2A98B9         NEXT:    LD    HL,(LINKNO)      次の行の先頭アドレスを取り出してL0へ飛ぶ
B94F C303B9                  JP    L0
                    ;
B952                REAWIN:
B952 7C                      LD    A,H              ポート70Hにオフセット値を設定して必要な部分をウィン
B953 D370                    OUT   (OPRT70),A       ドウから読めるようにする
B955 2680                    LD    H,SWINDW.R8      実際のテキストRAMアドレスをウィンドウ内のアドレスに
B957 C9                      RET                    変換する
                    ;
B958 5E             GETDE:   LD    E,(HL)
B959 23                      INC   HL
B95A 56                      LD    D,(HL)           HLのポインタが指すメモリの内容を2バイトDEに取り出
B95B 23                      INC   HL               す
B95C C9                      RET
                    ;
B95D                BIASD2:
B95D F5                      PUSH  AF
B95E D5                      PUSH  DE               10000の位(10進)の数を次めCRTに出力する
B95F 111027                  LD    DE,10000
B962 CD7EB9                  CALL  BAD1             1000の位(10進)の数を求めCRTに出力する
B965 11E803                  LD    DE,1000                                                    行番号を出
B968 CD7EB9                  CALL  BAD1             100の位(10進)の数を求めCRTに出力する        力するルー
B96B 116400                  LD    DE,100                                                     チン
B96E CD7EB9                  CALL  BAD1             10の位(10進)の数を求めCRTに出力する
B971 110A00                  LD    DE,10
B974 CD7EB9                  CALL  BAD1             1の位(10進)の数を求めCRTに出力する
B977 7D                      LD    A,L
```

```
B978 CD87B9             CALL BAD4
B97B D1                 POP  DE
B97C F1                 POP  AF
B97D C9                 RET
                 ;
B97E AF          BAD1:  XOR  A
B97F ED52        BAD2:  SBC  HL,DE
B981 3803               JR   C,BAD3
B983 3C                 INC  A
B984 18F9               JR   BAD2         それぞれの位の値を求め，アスキー・コードに変換して出力する
                 ;                        BAD3のADD HL,DEは，引きすぎた数を戻すために必要
B986 19          BAD3:  ADD  HL,DE
B987 C630        BAD4:  ADD  A,30H
B989 CD0D3E             CALL PUTCHR
B98C C9                 RET
                 ;
B98D 3E0D        CRLF:  LD   A,0DH
B98F CD0D3E             CALL PUTCHR
B992 3E0A               LD   A,0AH        CR, LFを出力して改行する
B994 CD0D3E             CALL PUTCHR
B997 C9                 RET
                 ;
B998             LINKNO:DS   2            リンク・ポインタ（次の行の先頭アドレス）の保存エリア
B99A             LINENO:DS   2            行番号の保存エリア
                 ;
B99C                    END
```

リスト22は，行番号の若いものから順に読み出し，その行がREM文(注釈行)であれば行番号とその内容をCRTに表示するプログラムで，REM文でなければ無視します．なぜREM文のみの表示を行うかといいますと，他のステートメントではそれぞれの中間言語の形に変換されてしまうからです．このプログラムでは，行番号に続いてREMか〝'〟の中間言語があるかどうかを調べています(図６.22)．注釈行であれば，保存しておいた行番号と内容を表示して，次の行を調べます．行番号は２バイトの２進数で格納されていますから，BIASD2のルーチンで，５桁の10進数に変換して出力しています．不用な０の省略は行っていません．

命令		中間言語
〝REM〟	→	8F
〝'〟	→	3A 8F E9

図6.22　注釈命令の中間言語

テキストの先頭アドレスはN_{88}-BASICのワークエリアから得ています．このプログラムは，リンク･ポインタの値が０のとき，つまりテキストの終わりを検出したら終了してモニタに戻ります．

この節のテーマであるI/Oポート関係は，REAWINというルーチンに現れています．これは，実際のテキストRAM上のアドレスを与えるとウィンドウにその部分が現れるようにして，そのウィンドウ上のアドレスを返すルーチンです．

6.6 文字列のサーチ

サーチ，すなわち検索の例として，テキスト・エリアにあるBASICプログラム中から任意の文字列を探すプログラムを考えます．

ここでは，最も単純な例として1文字ずつを比較して一致するか否かを調べていく方法を使います．

BASICのプログラムという，どのようなデータが格納されているのかわからないプログラムの場合には，この方法が一般的であると思います．ところが，サーチするデータの型式および構造が一定であるような場合には，もっと効率のよいサーチの方法もあります．

BASICのプログラムが格納されるテキスト・エリアは通常選択されていないため，**6.5 I/Oポートを使う**の節で説明した，テキスト・ウィンドウを通してのぞく，あるいはRAMバンクを選択して直接読むのどちらかを行わなければなりません．ここでは後者の方法を使って1文字ずつを比較し，一致すればその行番号を表示するようにしてあります．行番号は2バイトの2進データで格納されていますから，それを10進数に変換して出力します．

RAMバンク，ROMバンクの切り換えは，出力ポート31Hのビット1を操作します．ポート31Hは，入力と出力で意味が異なりますので，N_{88}-BASICでは，出力ポート31Hへ出力したデータをE6C2H番地に保存してありますから，それを使用して他のビットに影響を与えないようにします．RAMが選択されているときには，内部ルーチンのコールはもちろんできません．また，割込みは禁止しておかなければなりません．

さて，リスト23で1文字ごとの比較をしているのは，B925H番地からのルーチンです．ここでは比較とともに文字列の終わりと行の終わりを調べています．

B928H番地でHLレジスタ・ペアを保存しているのは，文字列の途中で不一致が検出された場合に，次の検索を始める位置を設定するためです．

BASICのプログラムは中間言語の型式でテキスト・エリアにありますから，LISTで出力される文字列がそのままサーチできるとは限りませんが，REM文であればその内容はすべてそのままのイメージでサーチすることができます．

リストは少し長めですが，多くのルーチンはすでに説明したものですから，そんなに難しいことはありません．

リスト23

```
                    ;       SEARCH STRINGS1
                    ;       3/22    BY M.F
                    ;
3583                GETCHR:EQU  3583H        キー入力ルーチンのエントリ・アドレス
3E0D                PUTCHR:EQU  3E0DH        CRT出力ルーチンのエントリ・アドレス
35CE                INKEY: EQU  35CEH        リアルタイムキー入力ルーチンのエントリ・アドレス
E658                TXTSTA:EQU  0E658H       テキストスタート・アドレスの格納エリア
0031                OPRT31:EQU  31H          CRTとメモリの設定を行う出力ポート
E6C2                DAT31: EQU  0E6C2H       出力ポート31Hに出力したデータが保存されているエリア
```

```
5F92            LINPT1:EQU   5F92H          ライン入力ルーチンのエントリ・アドレス
E9B9            DATBUF:EQU   0E9B9H         キー入力バッファの先頭アドレス
0013            CTRLS: EQU   13H            CTRL-Sのアスキー・コード
0003            CTRLC: EQU   03H            CTRL-C (STOP) のアスキー・コード
                ;
                       ORG   0B900H
B900            START:
B900 CDA6B9            CALL  INPUT          }ライン入力を行い,中断およびデータがなければEXITへ
B903 387D              JR    C,EXIT         }飛ぶ
B905 CDBEB9            CALL  CRLF
B908 2A58E6            LD    HL,(TXTSTA)    テキストのスタート・アドレスを取り出す (通常0001H)
                ;
B90B F3         RAMSEL:DI                   |
B90C 3AC2E6            LD    A,(DAT31)      |ポート 31 Hにビット1をセットしたデータを出力してRA
B90F F602              OR    2              |Mを選択する
B911 D331              OUT   (OPRT31),A     |
                ;
B913 CDA1B9     GETTXT:CALL  GETDE          |
B916 7B                LD    A,E            |リンク・ポインタをDEに取り込み
B917 B2                OR    D              |テキストの終わりであればEXITへ飛ぶ
B918 2868              JR    Z,EXIT         |
B91A ED53D7B9          LD    (LINKNO),DE    |
B91E CDA1B9            CALL  GETDE          }リンク・ポインタと行番号を保存する
B921 ED53D9B9          LD    (LINENO),DE    |
                ;
B925 11B9E9     COMPE1:LD    DE,DATBUF      }キー入内バッファ内のポインタを設定し，保存する
B928 22DBB9            LD    (TEMP),HL      }
B92B 1A         COMPE2:LD    A,(DE)         |
B92C B7                OR    A              }データの終わりであれば,同じ文字列であるからFINDへ飛ぶ
B92D 2812              JR    Z,FIND         |
B92F 47                LD    B,A
B930 7E                LD    A,(HL)         |
B931 B7                OR    A              |行の終わりであれば発見できなかったとして,次の行を調べ
B932 2848              JR    Z,NEXT         |るため NEXTへ飛ぶ
B934 B8                CP    B              }一致していれば次の文字を較べるためCOMPE2へ飛ぶ
B935 2004              JR    NZ,COMPE3      }
B937 13                INC   DE             |
B938 23                INC   HL             }不一致であればCOMPE3へ飛ぶ
B939 18F0              JR    COMPE2         |
                ;
B93B 2ADBB9     COMPE3:LD    HL,(TEMP)      |
B93E 23                INC   HL             }ポインタを進めてCOMPE1へ飛ぶ
B93F 18E4              JR    COMPE1         |
                ;
B941 CD97B9     FIND:  CALL  ROMSEL
                ;
B944 2AD9B9            LD    HL,(LINENO)    |
                ;                           |
B947 010010            LD    BC,1000H       |
B94A 110000            LD    DE,0           |
B94D 29         BASD1: ADD   HL,HL          |
B94E 7B                LD    A,E            |
B94F 8F                ADC   A,A            |
B950 27                DAA                  |
B951 5F                LD    E,A            |
B952 7A                LD    A,D            |
B953 8F                ADC   A,A            |
B954 27                DAA                  |
B955 57                LD    D,A            |同じ文字列が発見されたらその行番号を10進5桁で出力す
B956 79                LD    A,C            |る
B957 8F                ADC   A,A            |
B958 27                DAA                  |
B959 4F                LD    C,A            |
B95A 10F1              DJNZ  BASD1          |
                   ;
B95C CD8EB9        ADHEX1:CALL  BCD2
B95F 4A                   LD    C,D
B960 CD86B9               CALL  BCDASC
B963 4B                   LD    C,E
B964 CD86B9               CALL  BCDASC
B967 CDBEB9               CALL  CRLF
                   ;
B96A CDCE35               CALL  INKEY
B96D 280D                 JR    Z,NEXT
B96F FE13                 CP    CTRLS
B971 2806                 JR    Z,STOP
B973 FE03                 CP    CTRLC            }出力中の停止，中断を行う
B975 280B                 JR    Z,EXIT
B977 1803                 JR    NEXT
B979 CD8335        STOP:  CALL  GETCHR
                   ;
B97C 2AD7B9        NEXT:  LD    HL,(LINKNO)      }次の行の先頭アドレスを取り出してRAMSELへ飛ぶ
B97F C30BB9               JP    RAMSEL
                   ;
B982 CD97B9        EXIT:  CALL  ROMSEL           }N88 BASIC-ROMを選択してモニタへ戻る
B985 FF                   RST   38H
                   ;
B986 79            BCDASC:LD    A,C
B987 0F                   RRCA
B988 0F                   RRCA
B989 0F                   RRCA
B98A 0F                   RRCA
B98B CD8FB9               CALL  BCD1             }BCDデータをアスキー・コードに変換して出力する
B98E 79            BCD2:  LD    A,C
B98F E60F          BCD1:  AND   0FH
B991 C630                 ADD   A,30H
B993 CD0D3E               CALL  PUTCHR
B996 C9                   RET
                   ;
B997 F3            ROMSEL:DI
B998 3AC2E6               LD    A,(DAT31)
B99B E6FD                 AND   0FDH             }ポート31Hにビット1をリセットしたデータを出力してR
B99D D331                 OUT   (OPRT31),A        OMを選択する
B99F FB                   EI
B9A0 C9                   RET
                   ;
B9A1 5E            GETDE: LD    E,(HL)
B9A2 23                   INC   HL
B9A3 56                   LD    D,(HL)           }HLのポインタが指すメモリの内容をDE2バイトに取り込
B9A4 23                   INC   HL                む
B9A5 C9                   RET
                   ;
B9A6               INPUT:
B9A6 CDC9B9               CALL  MESG1C           }メッセージ“STRINGS?”を出力する
B9A9 20535452             DB    ´ STRINGS? ´,0
B9AD 494E4753
B9B1 3F2000
                   ;
B9B4 CD925F        IN1:   CALL  LINPT1
B9B7 D8                   RET   C
B9B8 23                   INC   HL
B9B9 7E                   LD    A,(HL)           }ライン入力をする
B9BA B7                   OR    A                 STOPで中断されるか，データがない場合はCYフラグを
B9BB C0                   RET   NZ                セットしてリターンする
B9BC 37                   SCF
B9BD C9                   RET
```

```
                ;
B9BE 3E0D       CRLF:   LD      A,0DH
B9C0 CD0D3E             CALL    PUTCHR
B9C3 3E0A               LD      A,0AH          CR，LFを出力して改行する
B9C5 CD0D3E             CALL    PUTCHR
B9C8 C9                 RET
                ;
B9C9 CDBEB9     MESG1C:CALL     CRLF
B9CC E3         MESG1:  EX      (SP),HL
B9CD 7E                 LD      A,(HL)
B9CE B7                 OR      A
B9CF 23                 INC     HL             メッセージ（文字列）出力ルーチン
B9D0 E3                 EX      (SP),HL
B9D1 C8                 RET     Z
B9D2 CD0D3E             CALL    PUTCHR
B9D5 18F5               JR      MESG1
                ;
B9D7            LINKNO:DS       2              リンク・ポインタの保存エリア
B9D9            LINENO:DS       2              行番号の保存エリア
B9DB            TEMP:   DS      2
                ;
B9DD                    END
```

6.7　グラフィック・パターンを動かす

この節では，マシン語でグラフィック画面を操作してみます．まず指定した座標に点を打つプログラムについて考え，次にそれを利用したグラフィック・パターンの表示プログラムを作ります．そして最後に，カーソル・コントロールキーによって，そのパターンを動かすプログラムについて説明しています．

6.7.1　点を打つ

グラフィック処理において最も基本的な操作は，指定した座標に指定した色の点を打つことでしょう．これは，BASICでいえばPSET文にあたります．

ただし，ここで紹介するプログラムはPSET文とは少し異なります．PSET文は，点を打つ前の色に関係なく指定した色で打ちますが，このプログラムでは，打つ前のカラー・コードと指定したカラー・コードの排他的論理和をとったものが表示されます．この方法は，バックのカラー・コードが0のときは指定した色が表示されますが，別のカラー・コードだと表示は違った色になってしまいます．その代わり，もう一度同じ座標に同じカラー・コードを打てばもとの色に戻るというメリットがあります．例えば，最初のカラー・コードが101，打つカラー・コードが011のときは

1回目　101　XOR　011＝110

2回目　110　XOR　011＝101

となります．これによって，点を打つプログラムで点を消し，前の状態に戻すことができます．

この方法はGlibのGカーソルの表示にも使われています.

■ PXORルーチン

では，プログラムについて説明しましょう．リスト24の最後のブロックPXORがこのプログラムです．座標はIXレジスタとIYレジスタにそれぞれx, y, Cレジスタにカラー・コードを入れてこのサブルーチンを呼び出せば，点のセットが行われます．

まず，リストの初めで座標の値のチェックを行います．これはそれぞれの最大値(xが639，yが199)以下であることのチェックのみを行っています．最大値を変えれば，表示領域の最大値は変えられますが，座標の最小値は常に(0，0)のままです．

次に，座標値からG-VRAMのアドレスとそのアドレス内のどのビットかを決定しなければなりません．G-VRAMは水平640ビット(80バイト)ですから，アドレス，ビットは次のようになります．

〈アドレス〉＝〈G-VRAM先頭アドレス〉＋〈y座標〉＊80＋(〈x座標〉￥8)
〈ビット〉 ＝〈x座標〉MOD 8
＝〈x座標〉AND 07H

80倍の乗算はMUL80というサブルーチンにより行っています．これは一般の乗算ではなく，2倍演算の組み合わせ(HL＊80＝HL＊64＋HL＊16)によって行っています．8で割る整数除算は右シフト3回で行います．

〈ビット〉で指定されたビットのみが1になっている1バイト・データと〈アドレス〉で指定されるG-VRAM中の1バイトとをXORすることにより，指定したドットのみを反転させることができます．これをカラー・コードで1がセットされているバンクに対して行うことによって，先に述べた操作ができます．

では，各バンクについて操作を行う方法について説明しましょう．まずカラー・コードを右シフトし，最下位ビットをCYフラグに移します．このとき，CYフラグが1なら必要なバンクに切り換え，指定されたアドレスと排他的論理和をとり，またCYフラグが0なら，この処理をスキップします．この処理を3回繰り返せばよいわけです．なお，この処理の間は割込みを禁止しておかなければなりません．

6.7.2 パターンを表示する

点を打つプログラムの次に適当な大きさを持ったパターンを表示する，BASICのPUT@文に相当するプログラムを考えてみましょう．

まず最初に思い浮かぶのが，点を打つ座標とカラー・コードを並べておき，端から点を打っていくという方法です．しかし，これではデータが大きくなってしまい，あまりよい方法とはいえません．そこでこのプログラムでは，表示領域を四角形に区切り，領域の大きさとカラー・コー

ドのみ与えるという方法をとりました.

■ データ構造

まず，データの構造について説明しましょう．先頭2バイトが領域の横，縦の大きさをドット単位で指定します．その後に，カラー・コードのデータが並んでいますが，カラー・コードは3ビットしかありませんから，1バイトに2個ずつ上位4ビットと下位4ビットに納めています．カラー・コードは，領域の左上から右方向に順に並んでおり，1行終われば1つ下の行へ移り，再び左側から並んでいきます．また，1バイト中に2個納まる順番は，上位4ビットが左寄り，下位4ビットが右寄りのデータになっています．パターン1組のデータの大きさは，横xドット，縦yドットとすると，

〈大きさ〉＝（x＊y＋1）￥2＋2　（バイト）

となります.

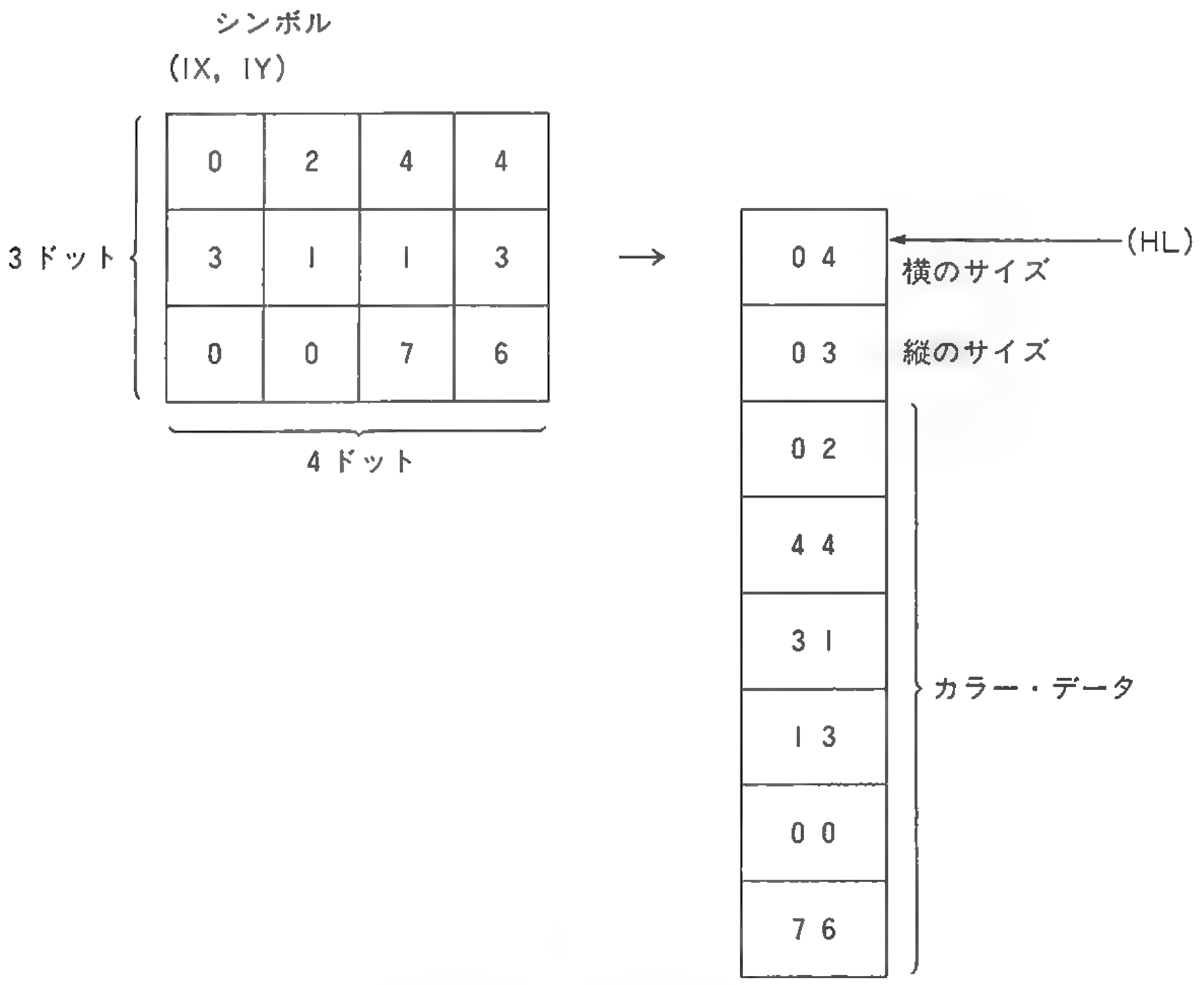

シンボル
(IX, IY)
0 2 4 4
3 1 1 3
0 0 7 6
3ドット
4ドット
→
04
03
02
44
31
13
00
76
(HL)
横のサイズ
縦のサイズ
カラー・データ

図6.23　PUTのデータ

■ PUT　SYMBOL　ルーチン

では，プログラムについて説明しましょう．このプログラムはリスト24のPUT SYMBOLという部分です．

このプログラムは，座標とデータの先頭アドレスを渡す必要があります．座標(パターン左上のx座標とy座標)はIXとIYに，データの先頭アドレスはHLに納めて渡し，プログラムは，この座標

とデータをもとに点を打つ座標とカラー・コードを設定してサブルーチンPXORを呼び出します．

最初に，カラー・コードのデータを１つ取り出します．これが何番目のデータであるかはDOFS納められており，そのアドレスは，データの先頭アドレスにDOFSETを２で割ったものを加算した結果となります．

点を打つ座標は，基点座標（パターンの左上）とそれに対する相対位置（X，Y）を加算することにより得られます．

サブルーチンCNVTが以上の処理を行い，その後サブルーチンPXORを呼び出せば点が打てます．

相対位置Xは点を１つ打つごとにインクリメントし，横のサイズを超えると０に戻り，Yをインクリメントします．これを繰り返し，Yが縦のサイズを超えたら終了となります．

６．７．３　パターンを動かす

ここでは，独立して動作するプログラムとして，カーソル・コントロールキーを使ってパターンを動かすプログラムを作りました．これは，ROM内のサブルーチンGETCHRを使ってキーボードからカーソル・コントロールキーを読み込み，その方向にパターンを１ドット移動するものです．

このプログラムは，まずキーボードから１文字読み込み，それがカーソル・コントロールキーであれば上下左右それぞれの処理に移ります．入力がスペースであれば処理を終了し，それ以外のキーは無視します．上下左右の処理はそれぞれ分かれていますが，行っている処理は似たようなものです．まず端か否かの判定を行い，端でなければ座標インクリメントまたは座標デクリメントを行い，端ならば何もしません．

各処理は先頭でDISPというサブルーチンを呼んでいますが，これは前に表示したパターンを消すためのものです．前にも説明した通りドットの表示は排他的論理和をとっていますから，表示している上に重ねて表示すれば前の図形を消すことができます．具体的には，キー入力ループの直前の呼出しが表示，各処理の先頭での呼出しが消去のためのものです．サブルーチンDISPは，座標とデータの先頭アドレスをセットして，サブルーチンPUTを呼ぶためだけのものです．

このプログラムはサブルーチンになっていますから，それを呼び出すメインルーチンが必要です．リスト24の最初の数行がそれにあたり，データの先頭アドレス（ここではC000H番地から）をセットし，このサブルーチンを呼んでいます．

実行してみるとわかりますが，このプログラムの速度はマシン語で書かれている割にはたいへん遅くなってしまいました．これは，パターンを描くためのデータをドット単位で扱っているためで，横方向８ドットを１バイトに納めるとかループを展開するなどすれば，かなり速くなるはずです．

リスト24

```
                                                      C000 → X方向のドット数
                                                      C001 → Y方向のドット数
                                                      C002 → カラー・コード
                 ;                                          (1バイトで2色)
                 ;       TEST PROGRAM
                 ;
                         ORG   0BC00H
BC00 2100C0              LD    HL,0C000H      }データの先頭アドレスをSYMSTにロードする
BC03 22E6BC              LD    (SYMST),HL     }
BC06 CD0ABC              CALL  MOVE         サブルーチンMOVEを呼ぶ
BC09 FF                  RST   38H
                 ;
                 ;       MOVE SYMBOL
                 ;
                 ;               by Machine Friend
                 ;
3583             GETCHR:EQU    3583H        キーボードの1文字入力待ちを行うインタプリタ内のサブルーチン・アドレス
001E             UP:     EQU   1EH          ↑のアスキー・コード
001F             DOWN:   EQU   1FH          ↓のアスキーコ・ード
001D             LEFT:   EQU   1DH          ←のアスキー・コード
001C             RIGHT:  EQU   1CH          →のアスキーコード
0020             SPACE:  EQU   ' '          スペースのアスキー・コード
                 ;       * REGISTER SAVE
BC0A F5          MOVE:   PUSH  AF              }
BC0B D5                  PUSH  DE              }
BC0C E5                  PUSH  HL              }各レジスタの退避する
BC0D DDE5                PUSH  IX              }
BC0F FDE5                PUSH  IY              }
                 ;
BC11 2ADABC              LD    HL;(XBIGIN)     }XX=320にセットする
BC14 22D6BC              LD    (XX),HL         }
BC17 2ADCBC              LD    HL,(YBIGIN)     }YY=100にセットする
BC1A 22D8BC              LD    (YY),HL         }
                 ;       * SYMBOL MOVE LOOP    サブルーチンDISPを呼ぶ
BC1D CDC7BC      MVLP:   CALL  DISP
BC20 CD8335      KEYLP:  CALL  GETCHR          サブルーチンGETCHRを呼ぶ
BC23 FE1E                CP    UP              }もし↑が押されたらSYMVPへ飛ぶ
BC25 2818                JR    Z,SYMUP         }
BC27 FE1F                CP    DOWN            }もし↓が押されたらSYMDWHへ飛ぶ
BC29 2858                JR    Z,SYMDWN        }
BC2B FE1D                CP    LEFT            }もし←が押されたらSYMLFTへ飛ぶ
BC2D 2832                JR    Z,SYMLFT        }
BC2F FE1C                CP    RIGHT           }もし→が押されたらSYMRGTへ飛ぶ
BC31 2872                JR    Z,SYMRGT        }
BC33 FE20                CP    SPACE           }もしスペースが押されてなかったらKEYLPへ飛ぶ
BC35 20E9                JR    NZ,KEYLP        }
                 ;       * EXIT
BC37 FDE1                POP   IY              }
BC39 DDE1                POP   IX              }
BC3B E1                  POP   HL              }レジスタの退避を解除する
BC3C D1                  POP   DE              }
BC3D F1                  POP   AF              }
BC3E C9                  RET                   リターンする
                 ;       * MOVE UP
BC3F CDC7BC      SYMUP:  CALL  DISP            サブルーチンDITPを呼ぶ（シンボルを消す）
BC42 2AD8BC              LD    HL,(YY)         }
BC45 ED5BE4BC            LD    DE,(YYMIN)      }HL=(YY)−(YYMIH)
BC49 AF                  XOR   A               }
BC4A ED52                SBC   HL,DE           }
BC4C 2009                JR    NZ,UPSKIP       もしYY≠0ならばUPSKPへ飛ぶ
BC4E 2AE4BC              LD    HL,(YYMIN)      }YY=0
BC51 22D8BC              LD    (YY),HL         }
```

```
BC54 C31DBC            JP    MVLP           MWLPへ飛ぶ
BC57 2AD8BC    UPSKIP:LD    HL,(YY)
BC5A 2B                DEC   HL            }YY=YY−1
BC5B 22D8BC            LD    (YY),HL
BC5E C31DBC            JP    MVLP          }MVLPへ飛ぶ
               ;        * MOVE LEFT
BC61 CDC7BC    SYMLFT:CALL  DISP            サブルーチンDISPを呼ぶ（シンボルを消す）
BC64 2AD6BC            LD    HL,(XX)
BC67 ED5BE0BC          LD    DE,(XXMIN)
BC6B AF                XOR   A             }HL=XX−XXMIN
BC6C ED52              SBC   HL,DE
BC6E 2009              JR    NZ,LFSKIP      XX≠0ならばLFSKPへ飛ぶ
BC70 2AE0BC            LD    HL,(XXMIN)    }XX=0
BC73 22D6BC            LD    (XX),HL
BC76 C31DBC            JP    MVLP           MVLPへ飛ぶ
BC79 2AD6BC    LFSKIP:LD    HL,(XX)
BC7C 2B                DEC   HL            }XX=XX−1
BC7D 22D6BC            LD    (XX),HL
BC80 C31DBC            JP    MVLP           MULPへ飛ぶ
               ;        * MOVE DOWN
BC83 CDC7BC    SYMDWN:CALL  DISP            サブルーチンDISPを呼ぶ（シンボルを消す）
BC86 2AD8BC            LD    HL,(YY)
BC89 ED5BE2BC          LD    DE,(YYMAX)
BC8D AF                XOR   A             }HL=YY−YYMAX
BC8E ED52              SBC   HL,DE
BC90 2009              JR    NZ,DWSKIP      YY≠YYMAXならばDWSKPへ飛ぶ
BC92 2AE2BC            LD    HL,(YYMAX)    }YY=YYMAX
BC95 22D8BC            LD    (YY),HL
BC98 C31DBC            JP    MVLP           MVLPへ飛ぶ
BC9B 2AD8BC    DWSKIP:LD    HL,(YY)
BC9E 23                INC   HL            }YY=YY+1
BC9F 22D8BC            LD    (YY),HL
BCA2 C31DBC            JP    MVLP           MVLPへ飛ぶ
               ;        * MOVE RIGHT
BCA5 CDC7BC    SYMRGT:CALL  DISP            サブルーチンDISPを呼ぶ（シンボルを消す）
BCA8 2AD6BC            LD    HL,(XX)
BCAB ED5BDEBC          LD    DE,(XXMAX)
BCAF AF                XOR   A             }HL=XX−XXMAX
BCB0 ED52              SBC   HL,DE
BCB2 2009              JR    NZ,RTSKIP      XX≠XXMAXならRTSKPへ飛ぶ
BCB4 2ADEBC            LD    HL,(XXMAX)    }XX=XXMAX
BCB7 22D6BC            LD    (XX),HL
BCBA C31DBC            JP    MVLP           MULPへ飛ぶ
BCBD 2AD6BC    RTSKIP:LD    HL,(XX)
BCC0 23                INC   HL            }XX=XX−1
BCC1 22D6BC            LD    (XX),HL
BCC4 C31DBC            JP    MVLP           MULPへ飛ぶ
               ;        * DISPLAY SYMBOL
BCC7 DD2AD6BC  DISP:   LD    IX,(XX)        IXに現在シンボルのある左上のX座標をロードする
BCCB FD2AD8BC          LD    IY,(YY)        IYに現在シンボルのある左上のY座標をロードする
BCCF 2AE6BC            LD    HL,(SYMST)     HLにシンボルのデータの入っている先頭アドレス・セット
BCD2 CDE8BC            CALL  PUT            サブルーチンPUTを呼ぶ
BCD5 C9                RET                  リターンする
               ;        * WORK AREA
BCD6           XX:     DS    2
BCD8           YY:     DS    2
BCDA 4001      XBIGIN:DW    320
BCDC 6400      YBIGIN:DW    100
BCDE 7F02      XXMAX: DW    639
BCE0 0000      XXMIN: DW    0
BCE2 C700      YYMAX: DW    199
BCE4 0000      YYMIN: DW    0
BCE6           SYMST: DS    2
```

```
                  ;
                  ;         PUT SYMBOL
                  ;                 IX      : X BASE POINT
                  ;                 IY      : Y BASE POINT
                  ;                 (HL)    : X SIZE
                  ;                 (HL+1)  : Y SIZE
                  ;                 (HL+2)  : DATA START
                  ;                           by Machine Friend
                  ;         * REGISTER SAVE
BCE8 D5           PUT:    PUSH DE
BCE9 C5                   PUSH BC              各レジスタを退避する
BCEA F5                   PUSH AF
                  ;
BCEB DD2265BD             LD   (XBASE),IX      IXの座標をXBASE(シンボルをPUTする基準)にロードする
BCEF FD2267BD             LD   (YBASE),IY      IXの座標をYBASE(シンボルをPUTする基準)にロードする
BCF3 7E                   LD   A,(HL)          XSIZEにシンボルの横幅をロードするをセットする
BCF4 326BBD               LD   (XSIZE),A
BCF7 23                   INC  HL              HL=HL+1
BCF8 7E                   LD   A,(HL)          YSIZEにシンボルの縦幅をロードするをセットする
BCF9 326CBD               LD   (YSIZE),A
BCFC 23                   INC  HL              HL=HL+1
BCFD 2269BD               LD   (DBASE),HL      DBASEにシンボルのデータの先頭アドレスをセットする
BD00 210000               LD   HL,0
BD03 226DBD               LD   (X),HL
BD06 226FBD               LD   (Y),HL          X, Y, DOFSETを0に初期化する
BD09 2271BD               LD   (DOFSET),HL
                  ;         * PUT SYMBOL
BD0C CD31BD       PUTLP:  CALL CNVT            サブルーチンCNVTを呼ぶ
BD0F CD73BD               CALL PXOR            サブルーチンPXORを呼ぶ
BD12 ED5B6BBD             LD   DE,(XSIZE)      E=XSIZE, D=YSIZE
BD16 3A6DBD               LD   A,(X)
BD19 3C                   INC  A               X=X+1
BD1A 326DBD               LD   (X),A
BD1D 93                   SUB  E               X=E (XSIZE) でなかったらPUTLPへ飛ぶ
BD1E 20EC                 JR   NZ,PUTLP
BD20 326DBD               LD   (X),A           X=0
BD23 3A6FBD               LD   A,(Y)
BD26 3C                   INC  A               Y=Y+1
BD27 326FBD               LD   (Y),A
BD2A 92                   SUB  D               Y=D (YSIZE) でなかったらPUTLPへ飛ぶ
BD2B 20DF                 JR   NZ,PUTLP
                  ;         * PUT END
BD2D F1                   POP  AF
BD2E C1                   POP  BC              各レジシタ退避を解除する
BD2F D1                   POP  DE
BD30 C9                   RET                  リターンする
                  ;
                  ;         SET DATA FOR PXOR  現在のシンボルのデータ (ループ回数ー1)
                  ;
BD31 2A71BD       CNVT:   LD   HL,(DOFSET)     HLにDOFSET(ループPUTLPをまわった回数)をロードする
BD34 CB3C                 SRL  H               HL/2 (シンボルを含むバイトの相対位置)
BD36 CB1D                 RR   L
BD38 F5                   PUSH AF              フラグ・レジスタを退避する
BD39 ED5B69BD             LD   DE,(DBASE)      DEにDBASE(シンボル・データの先頭アドレス)セットする
BD3D 19                   ADD  HL,DE           HL=HL+DE (データの絶対アドレス)
BD3E F1                   POP  AF              AFの退避を解除する
BD3F 7E                   LD   A,(HL)          Aにシンボルのデータをロードする
BD40 3804                 JR   C,CSKIP         HL (DOFSET) が奇数のときはCSKIPへ飛ぶ
BD42 1F                   RRA
BD43 1F                   RRA                  下位4ビットと上位4ビットを入れ換える
BD44 1F                   RRA
BD45 1F                   RRA
```

```
BD46 E607      CSKIP: AND  07H          A=A MOD 8 (カラー・コード)
BD48 4F               LD   C,A          Cにカラー・コードをロードする
BD49 2A71BD           LD   HL,(DOFSET)  ⎫
BD4C 23               INC  HL           ⎬DOFSET=DOFSET+1
BD4D 2271BD           LD   (DOFSET),HL  ⎭
BD50 DD2A6DBD         LD   IX,(X)       ⎫
BD54 ED5B65BD         LD   DE,(XBASE)   ⎬シンボルの1ドット分を指定する (X)
BD58 DD19             ADD  IX,DE        ⎭
BD5A FD2A6FBD         LD   IY,(Y)       ⎫
BD5E ED5B67BD         LD   DE,(YBASE)   ⎬シンボルの1ドット分を指定する (Y)
BD62 FD19             ADD  IY,DE        ⎭
BD64 C9               RET               リターンする
               ;       * WORK AREA
BD65           XBASE: DS   2
BD67           YBASE: DS   2
BD69           DBASE: DS   2
BD6B           XSIZE: DS   1
BD6C           YSIZE: DS   1
BD6D           X:     DS   2
BD6F           Y:     DS   2
BD71           DOFSET:DS   2
               ;
               ;        POINT EXCLUSIVE-OR
               ;               IX : X-AXIS
               ;               IY : Y-AXIS
               ;               C  : COLOR
               ;
               ;                          by Machine Friend
               ;
C000           GVRAM: EQU  0C000H
005C           GVRAM0:EQU  5CH
00FF           ERROR: EQU  0FFH
               ;       * REG.SAVE
BD73 E5        PXOR:  PUSH HL           ⎫
BD74 D5               PUSH DE           ⎬各レジスタを退避する
BD75 C5               PUSH BC           ⎭
               ;       * CHECK BOUNDARY
               ;       * X-AXIS
BD76 DDE5             PUSH IX           }HLにIXをコピーする              ⎫
BD78 E1               POP  HL                                            ⎪
BD79 ED5BE2BD         LD   DE,(XMAX)    DEに640をロードする              ⎪
BD7D B7               OR   A            }HL=HL-DE                        ⎪
BD7E ED52             SBC  HL,DE                                         ⎪
BD80 3050             JR   NC,OVER      もしHL≧640ならばOVERへ飛ぶ        ⎪
               ;       * Y-AXIS                                          ⎬座標内にあるか否か
BD82 FDE5             PUSH IY           }HLにIYをコピーする              ⎪についてテストする
BD84 E1               POP  HL                                            ⎪
BD85 ED5BE4BD         LD   DE,(YMAX)    DEに200をロードする              ⎪
BD89 B7               OR   A            }HL=HL-DE                        ⎪
BD8A ED52             SBC  HL,DE                                         ⎪
BD8C 3044             JR   NC,OVER      もしHL≧200ならばOVERへ飛ぶ        ⎭
               ;       * GET EFFECTIVE ADDRESS
BD8E FDE5             PUSH IY           }IYをHLにコピーする
BD90 E1               POP  HL
BD91 CDD8BD           CALL MUL80        サブルーチンMVL80を呼ぶ (HL=IY*80)
BD94 EB               EX   DE,HL        HLをDEにコピーする
BD95 DDE5             PUSH IX           }IXをHLにコピーする
BD97 E1               POP  HL
BD98 3E07             LD   A,7          ⎫
BD9A A5               AND  L            ⎬B=L  MOD 8 (HL/8の余り)
BD9B 47               LD   B,A          ⎭
```

```
BD9C CB3C          SRL   H             ┐
BD9E CB1D          RR    L             │
BDA0 CB3C          SRL   H             │HLは（HL／8）の整数解
BDA2 CB1D          RR    L             │
BDA4 CB3C          SRL   H             │
BDA6 CB1D          RR    L             ┘
BDA8 19            ADD   HL,DE         座標（IX, IY）のC000からの相対アドレス
BDA9 1100C0        LD    DE,GVRAM      DEにG-VRAMの先頭アドレス（C000）をロードする
BDAC 19            ADD   HL,DE         HL=HL+DE（HLは（IX, IY）の絶対アドレス）
              ;     * BIT PATTERN
BDAD 1680          LD    D,80H         Dに"10000000"をロードする
BDAF 78            LD    A,B           ┐
BDB0 B7            OR    A             │B（割り算の余り）が0ならばBLPENDへ飛ぶ
BDB1 2805     BROT: JR   Z,BLPEND      ┘
BDB3 CB1A          RR    D             DをCYフラグを含めて1ビット分各ビットを右へずらす
BDB5 3D            DEC   A             A=A−1
BDB6 18F9          JR    BROT          BROTへ飛ぶ
              ;     * PSET
BDB8 59       BLPEND:LD  E,C           Eにカラー・コードをロードする
BDB9 0E5C          LD    C,GVRAM0      CにG-VRAM選択ポートの先頭アドレス・セット
BDBB 0603          LD    B,3           繰り返し回数をセットする
BDBD F3            DI                  割込み禁止
BDBE CB3B     BLOOP: SRL E             カラー・コードの各ビットを右へ1ビット分ずらす
BDC0 3007          JR    NC,SKIP       CY=0ならSKIPへ飛ぶ
BDC2 ED79          OUT   (C),A         その色のVRAMを選択する
BDC4 7A            LD    A,D           }D(ビット・パターン)とHLの指すメモリの内容との
BDC5 AE            XOR   (HL)          }XORをとり，VRAMに出力する
BDC6 77            LD    (HL),A
BDC7 D35F          OUT   (GVRAM0+3),A  メインRAMを選択する
BDC9 0C       SKIP: INC  C             次のグラフィクVRAMを指定する
BDCA 10F2          DJNZ  BLOOP         Bから1ひいて0でなかったらBLOOPへ飛ぶ
BDCC FB            EI                  割込み許可
              ;     * NORMAL END
BDCD AF            XOR   A             A=0
BDCE C1            POP   BC            ┐
BDCF D1            POP   DE            │レジスタの退避を解除する
BDD0 E1            POP   HL            ┘
BDD1 C9            RET                 リターンする
              ;     * ABNORMAL END
BDD2 3EFF     OVER: LD   A,ERROR       A=FF
BDD4 C1            POP   BC            ┐
BDD5 D1            POP   DE            │レジスタの退避を解除する
BDD6 E1            POP   HL            ┘
BDD7 C9            RET                 リターンする
              ;
              ;     HL=80*HL
              ;
BDD8 29       MUL80: ADD HL,HL         ┐
BDD9 29            ADD   HL,HL         │DE=HL*16
BDDA 29            ADD   HL,HL         │
BDDB 29            ADD   HL,HL         ┘
BDDC 54            LD    D,H
BDDD 5D            LD    E,L
BDDE 29            ADD   HL,HL         }HL=HL*64
BDDF 29            ADD   HL,HL
BDE0 19            ADD   HL,DE         HL=HL+DE=80*HL
BDE1 C9            RET                 リターンする
              ;
              ;     WORK
              ;
BDE2 8002     XMAX: DW   640
BDE4 C800     YMAX: DW   200
BDE6               END
```

6.7.4　シンボル用のデータの作成

このプログラムのように，グラフィックス上にシンボルを表示するような場合，シンボル用のデータの作成という作業は不可欠です．この作業はたいへん楽しい(?)作業であるため，普通は作成用のBASICプログラムを使います．このようなプログラムは，『PC-8801 グラフィックのすべて』でも似たようなものが紹介されていますから参照されるのもいいですし，また自分で作っても難しいものではありません．

しかし今回は，あえてそのようなツールを利用せずにデータを起こしてみました（決して勧められることではありません）．

まず，縦横比が2対1のセクション・ペーパを用意します．この上に図形を描き，各ドットに対応するパレット番号を記入していきます．あとは，このデータをモニタで書き込むだけです．

このプログラム用のサンプル・データをリスト25に示します．これは32×12ドットの大きさのブルドーザです．横32ドットにしたのは理由があります．32ドットのデータは16バイトに納まるからです．これは，モニタのD，Eコマンドの表示幅と同じ，つまりセクション・ペーパ上のデータはそのままの形で画面に表示されるというわけです．ですから，最初に必要な領域に0を入れておけば，あとはカーソル・コントロールキーとテンキーだけでデータを作成できます．0から7までの数字とスペースだけでは形なんかわからないという人もいるかもしれませんが，心配は御無用，だいたいの形は右側の文字表示領域に現れ，眼のピントをぼかすと，不規則な文字の並びがブルドーザに見えてきます．うそだと思ったらやってみてください．

色がわからないのが欠点ですが，これで超簡易シンボル・エディタのできあがり(?)です．

リスト25

```
      ┌Xのサイズ
      │   ┌Yのサイズ
C000 [20] [0D] 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C010 00 00 00 00 00 00 00 00 00 00 07 00 00 00 00 00
C020 00 00 00 00 00 00 00 66 66 00 70 00 00 00 00 00
C030 00 00 00 02 00 04 00 06 60 00 70 00 00 00 00 00
C040 00 00 66 62 22 20 40 66 66 66 66 61 60 00 00 00
C050 00 00 66 62 22 20 40 66 03 03 00 06 10 00 00 00
C060 00 00 66 66 66 60 40 66 33 33 33 06 61 00 00 00
C070 00 00 66 66 66 66 66 66 66 66 66 66 60 10 00 55
C080 00 00 03 33 33 33 33 33 33 33 33 33 55 51 55 50
C090 00 00 33 44 40 00 40 00 00 40 05 55 33 00 55 50
C0A0 00 00 33 46 66 66 65 55 55 55 55 55 55 55 55 50
C0B0 00 00 33 44 40 40 04 00 40 04 04 44 33 00 55 55
C0C0 00 00 03 33 33 33 33 33 33 33 33 33 30 00 00 00
C0D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0E0 00 00
```

66，70，33……などは4ビットごとのカラー・コード

```
     ff p
      ` p
fb˚ @ffffa`
fb˚ @f
fff`@f333 a
ffffffffff`  U
 333333333UQUP
3D@ @  @ U3 UP
3FffeUUUUUUUUP
3D@@  @  D3 UU
 3333333330
```

なんとなくブルドーザに見えませんか？
“3”の文字はキャタピラです

6.8 内部ルーチンの解析

この章ではいくつかの内部ルーチンを紹介していますが，.これらは『Techknow』を参考にしています．

ところで，N_{88}-BASICにはまだ種々の演算やグラフィックスなどを操作する有用なルーチンがたくさんあるはずです．しかし，NECはインタプリタの内部について発表していませんから，これらのルーチンを利用するには，『Techknow』などを参考にして自分で内部ルーチンの解析を行う必要があります．

内部ルーチンの解析は，マシン語プログラムの動作を調べることですから，逆アセンブルを行います．幸いにも，PC-8801のモニタにはミニ逆アセンブラ機能がありますから，短いルーチンであれば，これで，充分解析していくことができます．このミニ逆アセンブラは，8080AからZ80(μPD780)になって拡張された命令のすべてを扱うことはできませんが，BASICインタプリタのルーチンの解析にはほとんど問題なく使用できます．

また本格的にインタプリタの内部構造を解析をしたい方は，DUAD-88Dの逆アセンブラ(Daisy)を使用すればよいでしょう(第7章参照)．

6.8.1 解析のKnow-how

内部ルーチンの解析においては，その動作はもちろんのこと，次のような点をおさえるようにします．

(1) 引数がある場合，それをどのように与えるのか

レジスタやメモリにどのような値を設定してコールするのか，ということです．

(2) 内容が変更されるレジスタについて

そのルーチンの入口と出口で，内容の変化するレジスタとそうでないレジスタを知る必要があります．

(3) そのルーチンからどのように情報が戻ってくるのか

ルーチンの実行結果がどのように与えられるかということです．フラグでその情報が与えられることもあります．

(4) エラーが起ったとき，どのような処理がなされるのか

不適当な引数などが与えられた場合，それを検出しているか否か，また検出した後どのような処理を行っているのかということです．

(5) 割込みに関すること

主に割込みの禁止と許可に関することです．禁止状態でなければならないのに，ルーチン内で許可(EI)されてしまう場合があります．

6.8.2 解析の実際

では実際に内部ルーチンの解析を行ってみましょう．例として，PC-8801のテキスト画面の座標とテキストVRAMアドレスの変換ルーチンを取り上げます．

このルーチンは，N$_{88}$-BASIC ROMの429DH番地から始まっていることが『Techknow』（付録2：N$_{88}$-ROM BASICインタプリタ解析）を参照するとわかります．表6.1は，『Techknow』からこのルーチンに関係する部分を引用させていただいたものです．

アドレス	項目名	機能
03B3	エラー出力	Eレジスタにエラー・コードを入れて分岐するとエラーが出力される
4257 ∫ 428A	VRAMアドレス	VRAM上の各行の先頭アドレス－VRAMTOPの値が格納されている
429D	VRAMアドレス計算	カーソル位置(H, L)→カーソル・アドレス(HL) カーソル位置は1～
431E	ライン・アドレス	Entry：L←テキスト画面　行#(1～) Exit　：DE＝行の先頭アドレス

表6.1　資料1（Techknowより引用）

まず先頭アドレス429DH番地からミニ逆アセンブラを用いて逆アセンブルしていきます．このとき，無条件ジャンプ(JP,JR)や無条件リターン(RET)などのプログラムの区切りとなる命令に注意して，リストをみていきます．これは，このルーチンの終わりを探すためと，無駄なリストをなるべくとらないようにするためです．リスト1では，42B3H番地のRET命令で区切れていることがわかり，またルーチンの終わりではないかと見当がつきます．

リスト26

```
                          ┌──┬─┐──────インテルニーモニック
429D    D5                PUSH D
429E    E5                PUSH H
429F    CD 431E           CALL 431E   各行の先頭VRAMアドレスDEに得る
42A2    E1                POP  H
42A3    6C                MOV  L,H     ⎫
42A4    26 00             MVI  H,00    ⎬横座標の値をHLに設定し直す
42A6    2D                DCR  L       ⎭
42A7    F5                PUSH PSW
42A8    3A EF89           LDA  EF89    ⎫
42AB    FE 29             CPI  29      ⎬横が41桁以上なら42B0Hへ飛ぶ(いわゆる80桁モード時)
42AD    30 01             JRNC 42B0    ⎭
42AF    29                DAD  H      横座標を2倍する（40桁モード時）
42B0    F1                POP  PSW
42B1    19                DAD  D      VRAMアドレスをHLに得る
42B2    D1                POP  D      429DHのPUSHDに対応している
42B3    C9                RET
42B4    2A 4289           LHLD 4289
42B7    EB                XCHG
42B8    2A E6C4           LHLD E6C4       (インテルニーモニックとμPD780の命令の)
42BB    19                DAD  D          (対応はインストラクション表を参照.      )
42BC    54                MOV  D,H
```

次に，途中でコールしている431EH番地からのルーチンを逆アセンブルしていきます．

リスト27

```
431E   E5          PUSH H
431F   7D          MOV  A,L     }縦座標が26以上であればエラー出力ルーチ }
4320   FE 1A       CPI  1A      }ンへ飛ぶ                               }縦座標は，1～25でな
4322   D2 0B06     JNC  0B06    }                                       }ければならない
4325   3D          DCR  A       }縦座標が0であればエラー出力ルーチンへ  }
4326   FA 0B06     JM   0B06    }飛ぶ                                   }
4329   CD 4333     CALL 4333    データテーブルを参照して各行に対するオフセット値を得る
432C   2A E6C4     LHLD E6C4    VRAMのスタートアドレス(0F3C8H)を取り出す
432F   19          DAD  D       }
4330   EB          XCHG         }行の先頭VRAMアドレスをDEに得る
4331   E1          POP  H       }
4332   C9          RET
4333   07          RLC          }
4334   6F          MOV  L,A     }データは2バイトずつであるから，2倍してHLに設定し直す
4335   26 00       MVI  H,00    }
4337   11 4257     LXI  D,4257  オフセットデータ・テーブルの先頭アドレスを設定する
433A   19          DAD  D       各桁に対応するオフセット値の格納されているアドレスを得る
433B   5E          MOV  E,M     }
433C   23          INX  H       }オフセット値をDEに取り出す
433D   56          MOV  D,M     }
433E   C9          RET
```

リスト28はB06H番地からのルーチンです．ここで3B3H番地へ無条件分岐していますが，これは表６．１によってエラー出力ルーチンとわかりますから，これ以上の追跡はやめます．

参照しているワーク・エリアについては，ほとんど『Techknow』に頼らなければなりません．必要な部分の引用を表６.２に示します．

リスト28

```
0B06   1E 05       MVI  E,05    }エラー・コードをEに設定してエラー出力ルーチンへ無条件ジャンプ
0B08   C3 03B3     JMP  03B3    }している
0B0B   7E          MOV  A,M
0B0C   FE 2E       CPI  2E
0B0E   ED          ???
0B0F   5B          MOV  E,E
0B10   AB          XRA  E
0B11   E6 CA       ANI  CA
0B13   0D          DCR  C
0B14   0A          LDAX B
0B15   CD 0B34     CALL 0B34
```

アドレス	ラベル	機能・用途
E6C4，C5	VRAMAD	テキスト画面アドレスTOP （初期値：F3C8）
EF89	LINWDT	画面横

表6.2　ワーク・エリア（Techknowより引用）

さて，ここまで資料が揃えば，あとはプログラムの流れを追いながら動作を考えていけます．この場合は，VRAMアドレス変換を行うルーチン（テキスト画面の座標）であることがわかっていますから，テキスト画面の構成に関する知識は当然必要です（『Techknow』参照）．

プログラムの細かい点はリストの注釈を参考にしていただくとして，このルーチンの仕事の要点だけを述べると次のようになります．

40桁モード
　V.ADRES＝&HF3C8＋120＊縦座標＋2＊横座標
80桁モード
　V.ADRES＝&HF3C8＋120＊縦座標＋横座標

この，&HF3C8＋120＊縦座標の部分は各行の先頭VRAMアドレスを得るところで，431EH番地からのルーチンに相当します．

431EH番地のルーチンではこのような計算は行わず，4257H番地から始まるデータ・テーブルを参照することによって，同じ機能を実現しています．

リスト29

```
4257 00 00 78 00 F0 00 68 01 E0 01 58 02 D0 02 48 03
4267 C0 03 38 04 B0 04 28 05 A0 05 18 06 90 06 08 07
4277 80 07 F8 07 70 08 E8 08 60 09 D8 09 50 0A C8 0A
```

データ・テーブルは2バイト単位で，それぞれの行の先頭アドレス－F3C8Hの値が格納されています．テーブルは4257H番地から428AH番地までですが，429DH番地からのルーチンでは4288H番地までのデータしか使いません．

ワーク・エリアのEF89H番地には，そのときにとり得る横座標の最大値が格納されています．普通は40(28H)か80(50H)となっていますが，プログラムなどで直接変更することによって横60桁などにすることができます．

B06H番地からはエラー・コード5 (Illegal function call)の出力ルーチンとなっていますが，ここではこのルーチンの詳しい動作は調べていませんので，このB06H番地へは，分岐しないようにします．この場合，引数の縦座標としては，1～25の値しか与えないように，429DH番地のルーチンをコールする前にチェックするようにすればよいでしょう．

では，ここで紹介した内部ルーチンの働きをまとめておきます．

アドレス　429DH
機　能　テキスト画面の座標からVRAMのアドレスを得る

説　明　Hレジスタに横座標，Lレジスタに縦座標を入れてコールすると，対応するテキストVRAMのアドレスを求めてHLレジスタ・ペアに格納して戻る．縦座標が1～25の範囲にない場合，〝Illegal function call〞エラーになる．横座標のチェックは行われないので，80以上を指定するとアトリビュート・エリアを破壊することがある

引　数　Hレジスタに横座標(1～80)，Lレジスタに縦座標(1～25)をセットする

レジスタ　A，F，H，Lが変更される

6.9　ランダム・テクニック

本章のしめくくりとしてランダム・テクニックの節を設けました．これらはマシン語のプログラミングをしていく上で役に立つと思われるテクニックです．

6.9.1　乱数の発生

BASICでゲーム・プログラムなどを作る際はRND関数で乱数を得ています．しかし，マシン語によるプログラムの場合，RND関数のように便利な機能はありませんから，なんらかのサブルーチンを作らなければなりません．

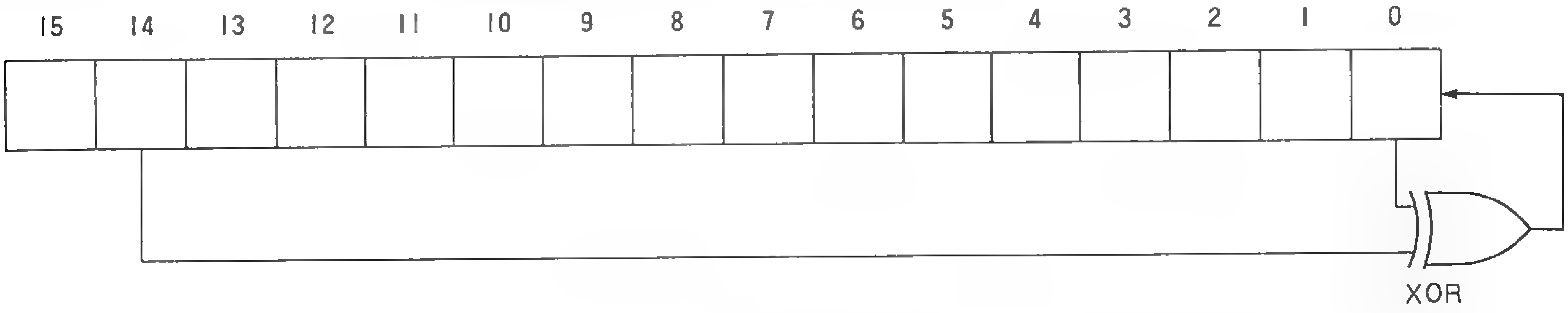
そこで，ここでは乱数を発生するサブルーチンについて紹介します．

■ M系列

コンピュータを使って，あるアルゴリズムにより生成する乱数は，正確な意味では乱数とはいえません．なぜなら，アルゴリズムに従って考えることによって次の値を知ることができるからです．そのため，プログラムで生成される乱数を擬似乱数といいます．

しかし，コンピュータで完全な乱数を作ることができないわけではありません．極端な話，コンピュータでマニピュレータ(ロボットの手のようなもの)を制御し，サイコロを振らせるなどすれば，本当の乱数が得られます．ここでは擬似乱数の一種であるM系列の乱数を発生させてみます．

適当な長さのレジスタを用意し，これを左にシフトします．シフトすると最下位ビットがあきますが，ここにはレジスタの中の任意の2ビットの排他的論理和を納めます．この任意の2ビットを変えることにより，同じ値が出てくる周期が変化します．



この場合ビット0とビット14のXORをとり，左シフト後のビット0とする

図6.24　M系列の発生

リスト30に，M系列で乱数を発生するプログラムを示します．

リスト30

```
                   ;
                   ;      TEST PROGRAM
                   ;
                          ORG   0BC00H
BC00 CD04BC               CALL  RAND          サブルーチンRANDを呼ぶ
BC03 FF                   RST   38H
                   ;
                   ;      RANDOM NUMBER
                   ;             by Machine Friend
                   ;
                   ;      * REGISTER SAVE
BC04 E5            RAND:  PUSH  HL           ⎫
BC05 D5                   PUSH  DE           ⎬各レジスタの内容を退避する
BC06 C5                   PUSH  BC           ⎪
BC07 F5                   PUSH  AF           ⎭
                   ;      * BIT TEST
BC08 2136BC               LD    HL,TBIT1      HLを0にロードする
BC0B CD28BC               CALL  SLRND         サブルーチン SLRHDを呼ぶ
BC0E F5                   PUSH  AF           ⎫DEにAFの内容をコピーする(Eのビット0←CY)
BC0F D1                   POP   DE           ⎭
BC10 2137BC               LD    HL,TBIT2      HLにTBIT2をロードする
BC13 CD28BC               CALL  SLRND         サブルーチンSLRHDを呼ぶ
BC16 F5                   PUSH  AF           ⎫BCにAFの内容をコピーする(Cのビット0 ←CY)
BC17 C1                   POP   BC           ⎭
BC18 79                   LD    A,C          ⎫CとEのXORをとる
BC19 AB                   XOR   E            ⎭
BC1A 1F                   RRA                 CYフラグも含めてBの各ビットを右へ1ビット分ずらす
BC1B 2A34BC               LD    HL,(RND)      HLにRNDをロードする(CYへXORの結果を移す)
BC1E ED6A                 ADC   HL,HL         HLの2倍にCYフラグを加える(CYを含むHLのシフト)
BC20 2234BC               LD    (RND),HL      RNDにHLをコピーする
                   ;      * EXIT
BC23 F1                   POP   AF           ⎫
BC24 C1                   POP   BC           ⎬各レジスタの内容の退避を解除する
BC25 D1                   POP   DE           ⎪
BC26 E1                   POP   HL           ⎭
BC27 C9                   RET                 リターンする
                   ;
                   ;      SHIFT LEFT (RND)    ( BIT 15 -> CARRY )
                   ;              ( HL ) : SHIFT COUNT
BC28 46            SLRND: LD    B,(HL)        BにHLで示したメモリの内容をロードする
BC29 3E10                 LD    A,16          Aに16をロードする
BC2B 90                   SUB   B             A=A−B
BC2C 2A34BC               LD    HL,(RND)      HLにRNDの内容をロードする
BC2F 47                   LD    B,A           BにAをロードする
BC30 29            SLLP:  ADD   HL,HL         HLを2倍する
BC31 10FD                 DJNZ  SLLP          B≠0ならSLLPへ飛ぶ
BC33 C9                   RET
                   ;      * WORK AREA
BC34 0100          RND:   DW    0001H
BC36 00            TBIT1: DB    0
BC37 0E            TBIT2: DB    14
BC38                      END
```

このプログラムでは排他的論理和をとるビットを，TBIT1とTBIT2で指定できます．

ではこのプログラムの動作について説明しましょう．まず，いちばん重要なのがRNDという16ビットのレジスタ(CPUのレジスタではない)で，プログラム実行後ここに乱数が納められます．

リスト30ではここは定数のようにみえますが，単に初期値を置いているだけです．

このプログラムで最も複雑なのは，2つのビットの排他的論理和をとることでしょう．まず，サブルーチンSLRNDで指定したビット数だけ左シフトし，必要なビットをCYフラグに移します．このままAFをPUSHし，別のレジスタ・ペアにPOPすることで，フラグをレジスタにコピーします．プログラムではCレジスタとEレジスタにコピーし，排他的論理和をとっています．次に右ローテイトを行い，結果をCYフラグに戻します．後はCYフラグを左から入れるシフトですが，これはADC　HL，HL命令で行っており，HL＋HLで左シフト，CYフラグの加算によりビット0にCYフラグが納まり，見事一命令で処理が完了します．

このサブルーチンは，呼ぶごとに乱数が更新されていきますから，参照する前か後に一度呼んでください．また，初期値を0000Hにすると変化しなくなりますから，初期値を設定するときは0000H以外の値にしなければなりません．なぜなら，0と0の排他的論理和は0で，シフトしても0000Hのままとなってしまうからです．

■ その他の乱数

M系列はきちんとしたアルゴリズムのある乱数でしたが，もっと簡単にでたらめな数値を得ることができます．

まず第1に，ROMを適当なアドレスから読み出すという方法があります．この方法ではむらのない乱数は無理ですが，ゲーム程度なら充分実用になります．

またZ80系のCPUの場合，もう1つ方法があります．それは，Rレジスタを読み出すという方法です．Rレジスタはダイナミック RAM用の7ビットのリフレッシュ・カウンタで，命令を読み込むごとにインクリメントしますから，読み出すごとにほぼでたらめな値が得られます．もっともRレジスタの値は周期的に増加しますから，命令をいくつ実行したかを調べることにより決定できます．

Rレジスタは簡単に読むことができ，予測もほとんどできないので便利なのですが，残念ながら7ビットしかありません．また続けて何回も読み込むと似た値しか得られない(最初の読み込みと次の読み込みの間隔が短いと，その間に少ししかインクリメントされない)ので，Rレジスタはゲームでの乱数の発生などに向いています．あるいは，M系列の初期値にするといった使い方も考えられます．

6.9.2　メモリの節約

一般に，マシン語プログラムはBASICのプログラムに較べコンパクトになりますから，メモリにプログラムが納まらないといったことはめったにありません．しかし，プログラム以外に大量のデータがある場合や大きなBASICプログラムと共に使う場合などはそういうことも起こるかもしれません．このようなときは，マシン語プログラムやデータをなるべく小さくする工夫をしなければなりません．ここでは，このメモリを節約する工夫について説明しましょう．

プログラムを小さくするためには，次の方法があります．

(a) アルゴリズムの最適化を図る
(b) 繰り返し処理をループにする
(c) 似たような処理をサブルーチン化する
(d) なるべく，ワーク・エリアをメモリ上にとらない
(e) 不用になったワーク・エリアを再使用する
(f) 等価な動作をする命令はコードの短い方を用いる

(a)，(b)，(c) は明らかな事ですが，(b)，(c) に関しては注意しなければならない点があります．ループでは初期値設定や終了判定，サブルーチンではCALLとRET，スタックなどが余計に必要なため，処理の回数が少ないときや短いときなどは，そのまま展開した方がメモリの節約になることがあります．

(d)は，データをなるべくレジスタ中に置いてメモリの使用量を少なくするということです．μPD780の場合，裏レジスタやインデックス・レジスタまで使えば，小さいプログラムならほとんどワーク・エリアを使わずに実行できます．ただし，裏レジスタを使うとプログラムがわかりにくくなるという欠点があります．

(e)は，独立したサブルーチン（サブルーチンA，BでAからBを呼ぶ，あるいはこの逆を行なわない関係にあるとき）であれば，それぞれのワーク・エリアは同時に使われませんからその領域を共用することができます．しかし，値を保存しなければならないエリアは共用できません．

(f)の等価な動作をする命令というのは，次のようなものです．

```
(1)LD    A，0  と XOR    A
(2)SLA   L
             とADD     HL，HL
   RL    H
```

(1)はAレジスタをクリアする動作で，LD A,0は2バイト，XOR Aは1バイトです．(2)は，HLレジスタ・ペアを左シフトする動作で，SLA LとRL Hが4バイト，ADD HL,HLは1バイトです．このように，同じ動作をする命令なら短い方を選ぶことによってメモリが節約できます．ただし，フラグの変化などが多少違うことがあるので注意が必要です．

データを短くするのはプログラムを短くするのに較べると難しいのですが，それでもいくつかの方法があります．

(a) データ構造の見直し
(b) 無駄な部分を詰める

データ構造の見直しというのは，固定長データを可変長データに変換したり，共通部分をくくり出したりといったことで，無駄な部分を詰めるというのは，連続したスペースをタブに置き換

えるとか，4ビット・データを1バイトに1つずつ納めていたのを2つずつ納めて，無駄をなくすといったことです．

しかし，このようにしてデータを小さくしても，それを処理するプログラムで，圧縮したデータをもとに戻す部分が大きくなり過ぎては無意味です．

このような工夫をしてもまだメモリが足りないというようなときは，メモリを広くするしかありません．まず思いつくのがG-VRAMです．これを使えば最大48Kバイトのメモリが使用できます．しかし，グラフィック機能が制限され，バンク切り換えを行わなければならないので，プログラムが複雑になります．

他に，8000H番地から始まる1Kバイトのテキスト・ウィンドウを使う方法があります．テキスト・ウィンドウというのは，0000H～7FFFH番地までの32KバイトのRAMを読み書きするための領域です．0000～7FFFH番地はBASICモードではROM領域のため，裏のRAMを読もうとしてもROMの内容しか読めません．そこで，ハードウェアによってRAMのアドレスを変換し，8000H番地からの1Kバイトを通してアクセスするようになっています．例えば，0000H～03FFH番地の1Kバイトをテキスト・ウィンドウに現れるように設定すると，8000H番地へのアクセスはハードウェアによりアドレスが変換され，実際には0000H番地に対して行われます．このとき，本来の8000H番地からの1Kバイトのメモリは変化しません．

テキスト・ウィンドウに現れるRAM領域の設定は，ポートに先頭アドレスを書き込むことによって行います．70H(I／Oポート)にSSという1バイトのデータを書き込むと，テキスト・ウィンドウには，SS00H番地～SS00H+3FFH番地までの1Kバイトが現れます．このとき先頭アドレスの下位8ビットは必ず00Hになります．

テキスト・ウィンドウの裏にある本当の8000H番地～83FFH番地の1KバイトのRAMは，BASICシステムが使っていませんから，ここをマシン語領域として使用することが可能です．この領域をRAMとして使うためには，SSに80Hを設定すればよいのです．すると，8000H番地～83FFH番地の1KバイトのRAMが8000H番地～83FFH番地のテキスト・ウィンドウに現れます．つまり，ごく普通のメモリとして使えるのです．ただし，SSを書き換える前にBASICで使っていたSSを保存し，BASICに復帰する前にもとに戻す必要があります．

一般に，プログラムを小さくすることと速度を向上させることは相反することなので，適当な妥協点をみつけることが大切です．

6.9.3　スピードについて

マシン語で書かれたプログラムは，BASICなどで書かれたものより高速処理が可能です．しかし，同じ処理をするプログラムでも，いろいろな違いにより実行時間に差が出てきます．この差はなぜ生じ，またどうすれば速くなるのでしょうか．

速度の差は，主にアルゴリズムの違いによって生じます．例えばバイトの乗算を，シフトと加算の繰り返しで実行するのと，必要な回数だけ加算を繰り返すのとでは，どちらが速いかは結果

をみずとも明らかでしょう．また同じアルゴリズムでも，プログラムの書き方によって差が生じます．一般にメモリーレジスタ間の転送より，レジスターレジスタ間の転送の方が速く実行できます．例えば，BCレジスタ・ペアをDEレジスタ・ペアに転送することを考えてみましょう．これには，次のような方法があります．

```
(a)  PUSH   BC
     POP    DE
(b)  LD     D, B
     LD     E, C
```

(a)はスタックを介しての，(b)はレジスタ間の転送です．プログラムはどちらも2バイトですが，(a)はスタックをメモリ上にとっているため，そのためのメモリ・アクセスが4回必要になり，プログラムの読み出しと合わせて6回メモリ・アクセスが行われます．それに対し，(b)はメモリを使わないので，メモリ・アクセスはプログラムの読み出しの際の2回だけです．

他にも，プログラムの実行時間を短くする方法がいくつかあります．サブルーチン，ループの展開がその代表的なものでしょう．サブルーチンは，CALLでメモリ・アクセスを5回，RETで3回行いますから，かなりの速度ロスになります．これを避けるためには，サブルーチンを使わずに，メイン・プログラムにその処理をすべて組み込めばいいのです．定数回のループも同様です．例えば，シフトを6回行うには，次のようなプログラムが一般的です．

```
      LD     B, 6
LOP : SLA    A
      DJNZ   LOP
```

ところがこのループは必ず6回まわるのですから，次のように書き換えることができます．

```
SLA   A
SLA   A
SLA   A
SLA   A
SLA   A
SLA   A
```

このようにすることによって，プログラムは大きくなりますが，ループのための命令がないため処理は速くなります．

以上のようなことをそのときの状況によって使い分ければ，プログラムは大きくなりますが，処理は高速になります．

N_{88}-BASICは割込みとDMAを常時使っています．これによってプログラムは中断されますか

ら，割込みとDMAを禁止すればその分実行終了までの時間は短くなります．

■ プログラムの実行時間

CPUはクロックによって動作が進行していきます．すべての命令は，完了するまでに必要なクロック数が決まっていますから，これを計算することによってプログラムを完了するまでの時間を求めることができます．クロック数は付録のインストラクション表に載っていますから，これを使ってプログラムの実行時間を考えてみましょう．

```
                  クロック数
       LD    B, 10     7
       XOR   A         4
LOOP : ADD   A, B      4
       DJNZ  LOOP      8／13
```

これは，１から10までの和を求めるプログラムです．DJNZが８と13になっているのは，ジャンプするとき13，しないときが８ということです．

〈クロック数〉$= 7 + 4 + (4 \times 10) + (13 \times 9 + 8)$
$= 176$

PC-8801のクロックは4MHzですから，１クロック・サイクルは250nsとなり，プログラムの実行時間は

〈実行時間〉＝〈クロック・サイクル〉×〈クロック数〉
$= 250\text{ns} \times 176$
$= 44\mu\text{s}$

となります．ただし，これはDMAや割込みがかからないときの値です．

6.9.4　割込みについて

割込み(インタラプト)というのは，ある処理の実行中に別の処理を割込ませて実行することです．例えば，仕事中に電話がかかってきたとしましょう．仕事中に電話がかかってくると，私たちは普通仕事を中断して電話に出，電話が終れば再び仕事を続けます．また，もしその電話が仕事を打ち切る連絡であれば，仕事は打ち切られます．

上の例では，仕事がCPUの実行中の処理，電話が割込み処理に相当します．そして上の場合と同様，割込み処理のプログラムの結果，もとの処理を続行したり，あるいはそれを打ち切って別の処理を開始したりするのです．

割込み処理で重要なのは，本来の処理の実行中に割込みのことをいっさい意識していない点で

す．上の例で考えれば，仕事をしている人は電話がかかってくることを気にせずに仕事をしているということです．CPUには，割込みの要求があれば処理の切り換えを行うためのハードウェアが備わっており，プログラムで調べることなしに割込み要求を検出することができます．

割込みは，いつ発生するかわからない要求を別の処理を実行しながら検出し，処理しなければならないときに使います．N$_{88}$-BASICのON KEY GOSUBやON COM GOSUBなどは割込みを使っています．通常はBASICのメイン・プログラムを実行し，KEY割込みなどがかかるとその処理ルーチンへ分岐し，要求に応じた処理を行います．処理終了後は中断していたメイン・プログラムを続行します．N$_{88}$-BASICでは，キー入力も割込みで処理しています．プログラムの実行中に押したキーの内容はキー入力バッファに納まっていて，プログラム終了後画面に表示されます．これは，キーを押すことによって割込みがかかり，本来の処理を中断してキー入力処理を行っているからで，もし割込みを使っていなければプログラム実行中のキー入力は無視されてしまいます．

N$_{88}$-BASICは，他にも表６．３のような割込みを使っています．

用　途	BASICでの使用
CMT RS-232C 受信割込み	可
画面終了割込み	可
リアルタイムクロック(1/600s)	可
ユーザ割込み	不可
〃	不可
〃	不可
FDD用リザーブ	可
FDD用リザーブ	可

表6.3　N$_{88}$-BASICの割込みテーブル表

■ マシン語レベルの割込み

マシン語プログラムでも割込みを使うことができます．ここではマシン語レベルでの割込み処理について説明しましょう．

■ 割込み処理

割込み処理ルーチンは，ある場所から分岐し，またもとの場所に戻るもので，この動作はサブルーチンとほぼ同じです．異なるのは，分岐の原因がプログラムではなくハードウェアによるものであるという点です．

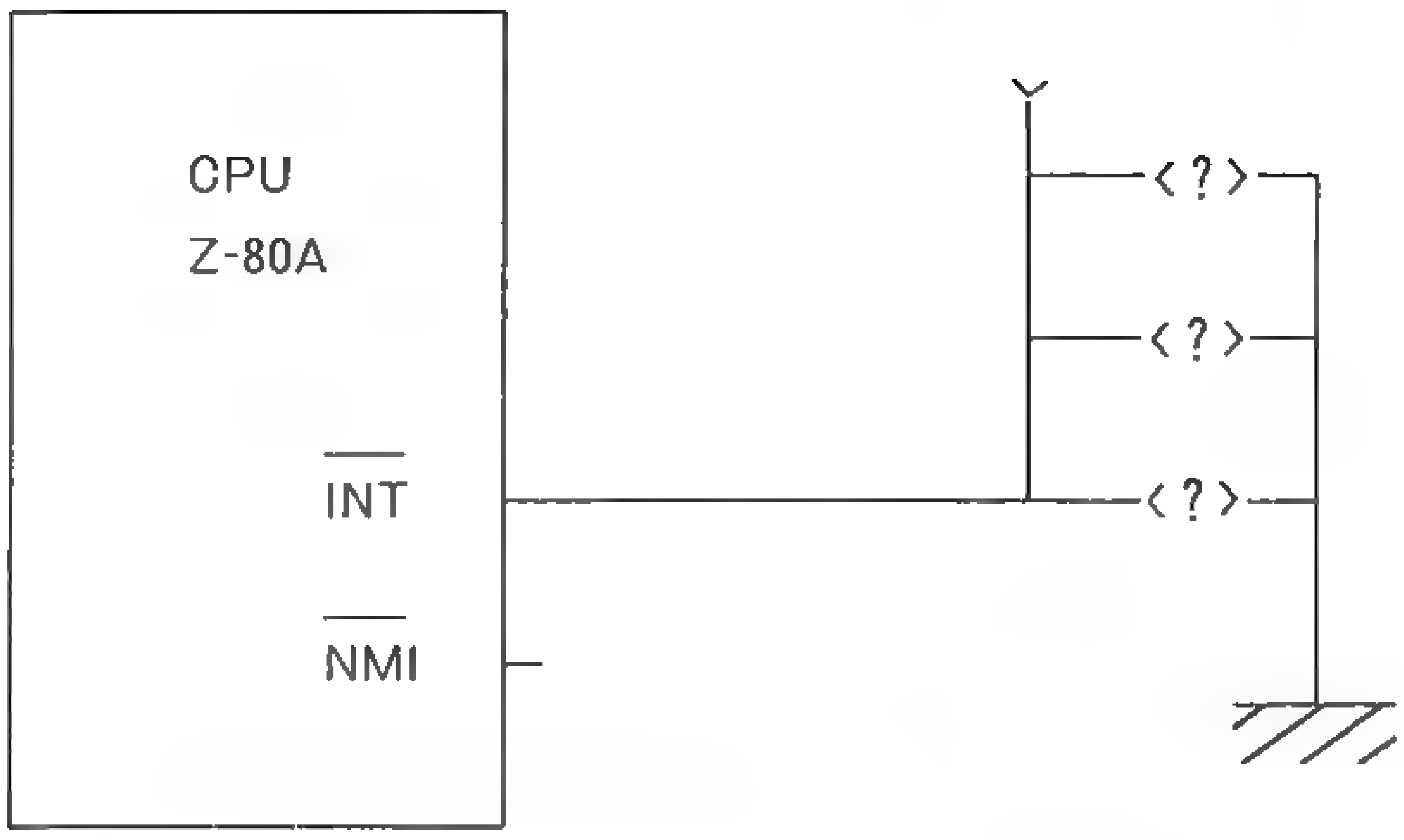


図6.25　割込み要因はハードウェアで決まる

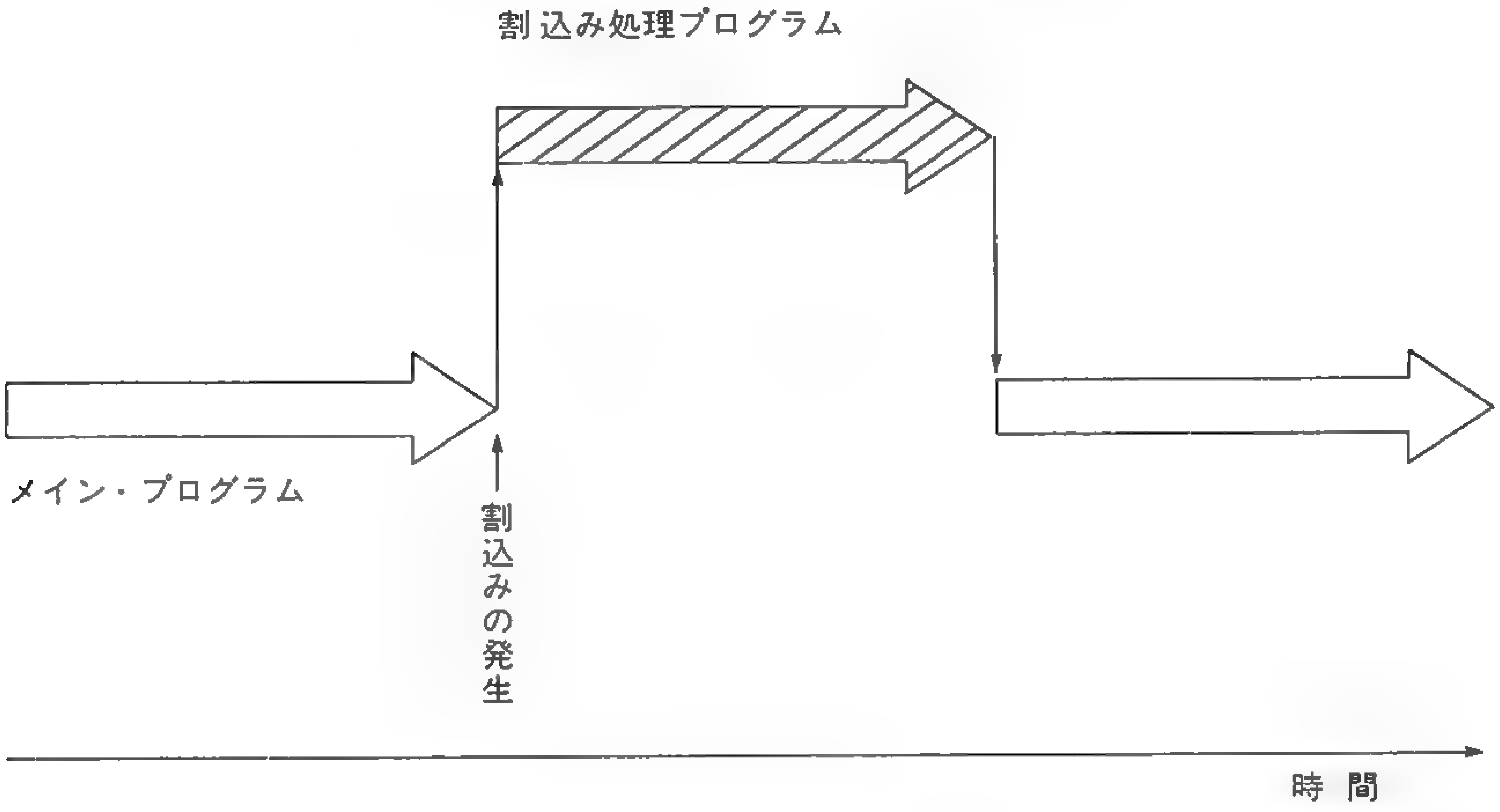


図6.26　割込み処理

■ 割込みモード

割込みモードとは，割込み要求を受け付けたときにどのように動作するかという指定です．μPD780では0から2までの3種類のモードがあります．この指定は，IM 0，IM 1，IM 2の3つの命令で行います．

モード0では，割込みを受け付けるとハードウェアによって送られてくる命令を実行します．このとき，普通はRST命令を使います．RST命令は1バイトのサブルーチンの呼び出し命令ですから，復帰アドレスをスタックに積んで指定されたアドレスへ分岐します．

モード1では，自動的にRST 38Hを実行します．

モード2は，ベクタ割込みというもので，最大128の割込みを区別することができます．ベクタ割込みというのは，割込みの発生要因(ハードウェア)がCPUに送るビット0が0になっている1バイトの割込み番号(00H～FEHの128種)によってその番号に対応する処理ルーチンへ，サブルーチンと同じように分岐するものです．番号からアドレスへの変換は図6.27のように行い，Iレジスタを上位バイト，番号を下位バイトとするアドレスから納められている2バイトが処理ルーチン

の開始アドレスとなるわけです．このアドレスの一覧表を割込みベクタ・テーブルといいます．N_{88}-BASICではこのモード２割込みを使っていて，割込みベクタ・テーブルはF300H番地から始まっています．他の割込みモードに変更することも可能ですが，そうすると，N_{88}-BASICの割込み（キー入力，RS-232Cなど）をサポートできなくなります．

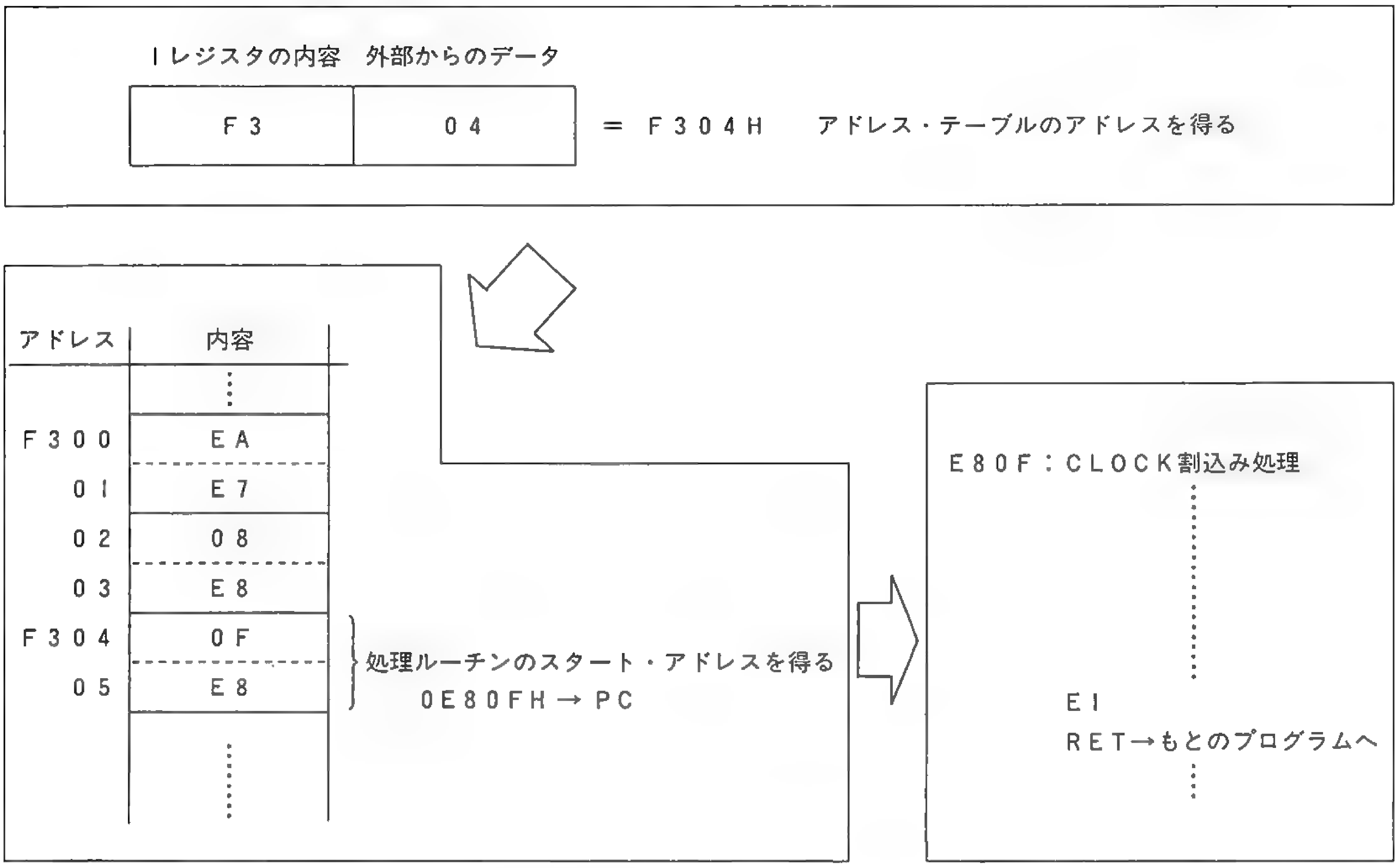

Ｉレジスタの内容
外部からのデータ
Ｆ３
０４
＝ Ｆ３０４Ｈ
アドレス・テーブルのアドレスを得る
アドレス
内容
Ｆ３００
ＥＡ
０１
Ｅ７
０２
０８
０３
Ｅ８
Ｆ３０４
０Ｆ
０５
Ｅ８
処理ルーチンのスタート・アドレスを得る
０Ｅ８０ＦＨ→ＰＣ
Ｅ８０Ｆ：ＣＬＯＣＫ割込み処理
ＥＩ
ＲＥＴ→もとのプログラムへ

図6.27　モード２の処理の流れ

■ 割込みの禁止

実行しているプログラムの内容によっては割込みをかけてほしくないときがあります．例えば，スタック・ポインタを操作しているときに割込みが発生すると，スタックの内容を破壊してしまいますし，バンクを切り換えていて割込み処理ルーチンがメモリ上に現れていないときは，割込みがかかっても行き先がありません．また割込みを禁止して実行時間を少しでも短くしたいときもあるでしょう．

割込みを禁止するときは，DIという命令を実行します．この命令の実行後は，割込み要求が発生しても無視します．ただし，要求があったということは記憶されるので，割込みを許可した後に受け付けます．

また割込みが発生したときも，以後の割込みが禁止されます．これは，同じ要因により何度も割込み処理ルーチンが起動されるのを防ぐためです．ユーザは，割込み処理が終了するか，次に割込みがかかってもよいと思われたら割込みを許可します．RETで中断していたプログラムへ復帰します．

割込みを許可するにはEIという命令を実行します．この命令を忘れると，次の割込みを受け付

けなくなりますから注意が必要です．

■ NMI（ノンマスカブル・インタラプト）

μPD780には，NMIの機能があります．NMIは，EI，DIにかかわらずに実行できる割込みで，一般には電源異常やハードウェアのエラー検出などに使われます．ただしPC-8801の標準システムではサポートされていません．NMIがかかると，プログラムは0066H番地に分岐します．

■ 割込みからの復帰

割込みから中断していたプログラムに復帰するには，RET命令を使います．μPD780には，他にRETI,RETNという命令もあります．RETIは外部に割込み処理ルーチンの終了を知らせる命令で，RETNはNMIの処理ルーチンの終了に使い，NMIがかかる前の状態に割込み許可フラグ（プログラムからはみえないハードウェア上のフラグ）を戻します．しかし，どちらの命令も，PC-8801では使う必要はありません．

■ 割込み処理ルーチン

割込み処理はプログラムのどこで発生するかは全く予測することはできず，また終了後はもとのプログラムを正常に続行させなければいけません．そのため，レジスタ類は完全に保存しなければなりませんし，また複数の割込みを扱うときは，優先順位の判定もしなければなりません．

本当は割込みのサンプルも示したかったのですが，ほとんどの割込みがBASICシステムにより処理されているため，適当な題材がなく，載せられませんでした．一般に，システムソフトが完備しているマシンほど，ユーザ・レベルでは割込みは使いにくくなってしまうのですが，割込みというのはたいへんおもしろい題材なので，興味のある人はいろいろやってみるのも楽しいでしょう．例えば，画面終了後割込みをうまく使用すると，グラフィック画面のチラつきを起こさずに描き込みができるようになります．

第7章
開発ソフトウェアについて

開発ソフトとは，プログラムを作成するための道具(ツール)としてのプログラムのことをいいます．例えば，PC-8801に備わっているN_{88}-BASICインタプリタは，BASIC言語のプログラムを作成するための開発ソフトです．またBASICインタプリタやアセンブラなどの言語処理プログラムだけではなく，ソース・プログラムを編集するためのエディタ(Editor)や，デバッガ(Debugger)およびファイルの管理などをつかさどるオペレーティング・システム(OS)なども開発ソフトであり，これらとハードウェアをも含めて開発システムと呼ぶこともあります．

ここではまず，PC-8801で使用できるマシン語プログラム開発のためのソフトとしてDUAD-88Dについて紹介し，次に多くの言語を使用できるCP／M(シーピーエム)を紹介します．

7.1　マシン語プログラム開発ソフト

マシン語プログラムを作成するには，前述したようにハンドアセンブルによるものとアセンブラを使う方法の2種類があります．

一口にアセンブラといっても，その機能は様々です．例えばPC-8801のモニタ・モードには，Aコマンドによるミニアセンブラがあります．しかし，これは1行に入力したニーモニックをすぐに変換してメモリにマシン語を生成していくものですから，ラベルが使用できず，またアドレス計算を自分でしなければなりません．またμPD780の命令をすべて使用できないことや，誤った入力をするとすぐにモニタのコマンド・レベルに戻ってしまうことなど，使いづらい点がいくつかあります．しかし，短いプログラムの作成には充分な機能でありハンドアセンブルと併用すれば効果的に使用できるでしょう．

7.2　DUAD-88D

では，次に本格的なアセンブラとしてASCII(アスキー)から販売されているDUAD-88D (Disk Utilities,Assembler,Dis-assembler,and Debugger for PC-8801:(以下DUADとする)を紹介します．

これは，PC-8801のためのマシン語プログラム開発応用ツールで，PC-8801ディスク・システムを用いて開発を行うものです．DUADは以下に示す機能のプログラムから構成されます。

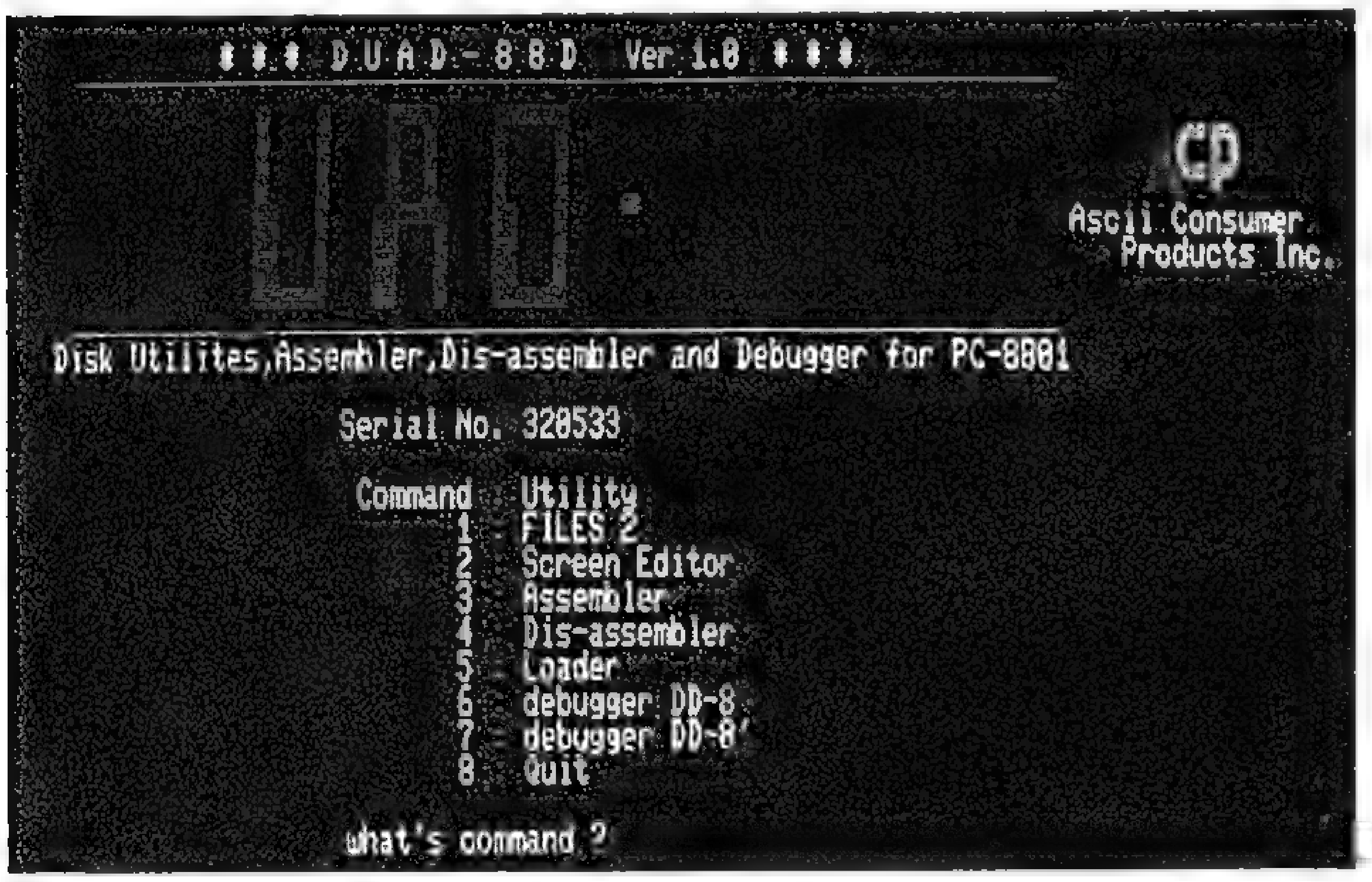
●Screen Editor	スクリーン・エディタ
●Assembler	アセンブラ
●Dis-assembler	逆アセンブラ
●Loader	ローダ
●Debugger	デバッガ

これらのプログラムはマシン語で記述されており，〝DUAD88〟というファイルネームのプログラムによって実行管理されます．

DUADのシステム・ディスクをドライブ１に入れて，

```
RUN  "DUAD88" ↵
```

を実行することにより，下の写真のように　メニュー形式でコマンド入力を促してきます．



```
### D U A D - 8 8 D  Ver 1.0 ###
                                        CP
                                   Ascii Consumer
                                     Products Inc.
Disk Utilites,Assembler,Dis-assembler and Debugger for PC-8801
              Serial No. 320533
              Command   Utility
                    1   FILES 2
                    2   Screen Editor
                    3   Assembler
                    4   Dis-assembler
                    5   Loader
                    6   debugger DD-8
                    7   debugger DD-8'
                    8   Quit
              what's command ?
```

DUAD-88Dのメニュー画面

ソース・プログラムを作成する場合には，２を入力しスクリーン・エディタを起動します．このエディタは，PC-8801のスクリーン・エディット機能を活かした操作性のよいものです．エディタではアセンブリ言語のソース・プログラムの作成，修正，および追加を行い，ディスク上にソース・ファイルを作ります．スクリーン・エディットモードでは，BASICのスクリーン・エディタにはなかった，文字列のサーチや置き換え，移動などの強力な編集機能がコントロール・キャラクタの入力によって使用できます．一度に編集できるソース・プログラムの大きさ(エディット・バッファ)は32Kバイトですが，ソース・ファイルのアペンドセーブができますから実際にはディスクの容量の許す限りのソース・ファイルを作ることができます．

ソース・プログラムの作成が終わったら，次にアセンブラを起動します．

DUADのアセンブラは，ザイログ社Z80の命令に準拠したニーモニックを扱う２パス形式のものです．２パスといのは，ソース・ファイルを２回読んでオブジェクトを生成することを表し，

１パスではラベル・テーブルを作成し，２パス目ではラベルを参照しながらオブジェクトを生成します．このアセンブラは，オブジェクト出力として，マシン語をHEXフォーマットと呼ばれる形式でファイルに出力します．HEXフォーマットとは，データを16進のアスキー・コードで表し，ロードするアドレス情報やチェックサムのコードなどをつけた形式です．従って，このオブジェクトは直接実行できません．

オプション	機　　　　能
l	アセンブル・リストの指定．指定がなければエラー行のみ表示
p	リストをプリンタに出力
o	オブジェクト・ファイルをディスク上に作成
f	リスト・ファイルをディスク上に作成
d	ラベル登録順に出力
s	ラベルをリードして出力
r	クロスリファレンス・リストを出力
h	d，s，rの出力をプリンタに指定
n	プリンタを改ページしてからの出力

表7.1　アセンブラ・オプションコマンド表

アセンブラで作成したプログラムを実行するためには，ローダを起動します．ローダは，オブジェクト・ファイルを読んで，必要があればオフセットを付けてメモリ上に実行可能なマシン語をロードします．オプション指定により即実行することができますが，BSAVEしておくと，それ以後はDUADのローダをいちいち起動しなくても使えますから便利です．

DUADのエディタ，アセンブラ，ローダを用いたプログラム開発の流れを図７．１に示します．

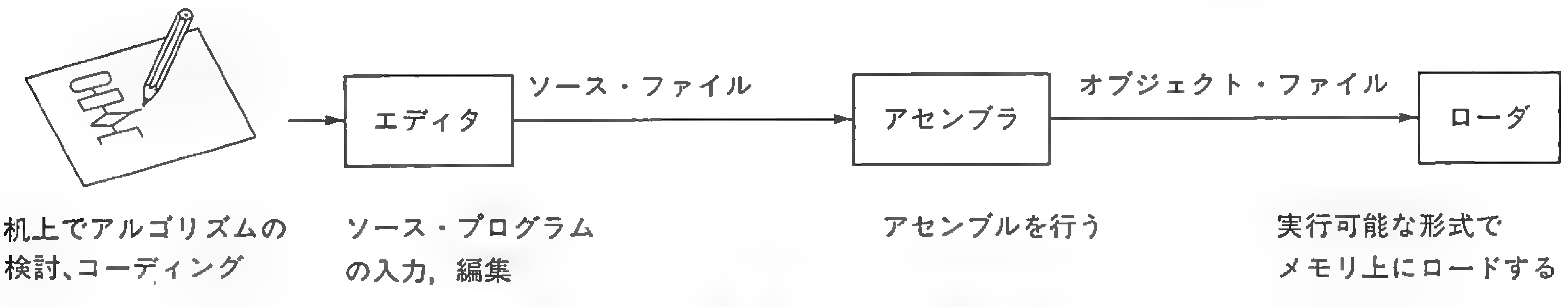

エディタ
ソース・ファイル
アセンブラ
オブジェクト・ファイル
ローダ
机上でアルゴリズムの検討、コーディング
ソース・プログラムの入力，編集
アセンブルを行う
実行可能な形式でメモリ上にロードする

図7.1　マシン語プログラム開発の流れ

これら３つのプログラムを用いれば，一通りマシン語プログラムを作成，実行をすることができます．しかし，作ったプログラムが一度で希望通りに動作することは稀であるため，デバッグを行うことになります．

アセンブラは，文法に関するエラーは発見してくれますが，プログラムの内容に関してはいっさい関知しません．短いプログラムなら，リストを見ながらレジスタの内容の変化を考えていけ

ば，たいていのバグは発見できますが，複雑なものになってくるとこの方法では手に負えません．

そこで，DUADにはデバッグ用のツールとして，DD-8，DD-8'デバッガが用意されています．

DD-8(Dynamic Debugger)は，機能的にはPC-8801のモニタをデバッグ向きにさらに拡張したものと考えることができます．ブレーク・ポイントの複数指定や回数指定，トレースなどの機能をうまく使うことにより，デバッグに要する時間を短縮することができます．また，1行ごとにアセンブルするアセンブラや逆アセンブラの機能は，ザイログニーモニックでZ80の全命令を扱うことができますから，簡単なプログラムはこのデバッガでも充分作成することができます．

コマンド	機能
Rxx，V	デバッガのロケーションを変更する
B，Bxx,K，Kxx	ブレーク・ポイントの設定，表示，解除をする
Gxx	ターゲット・プログラムの実行を行う
N，Nnn	ブレーク・ポイントのnn回通過の停止，スタート
T，Tnn	トレースを行う
S	レジスタ表示の形態変更
U，Uxx，C,Cxx	サブルーチンのトレースの表示，指定，解除を行う
X，Xr	レジスタの表示を行う
Dxx	メモリ内容をチェックサム付きで表示する
Axx	ニーモニックを用いてデータをセットする（アセンブラ）
L，Lxx	逆アセンブルを行う
P	プリンタ出力併用の切り換え
E，M	デバッガから抜ける

表7.2 デバッガコマンド

Daisy (Dis-Assembler，Interactive System) は，DUADの多機能逆アセンブラです．PC-8801のモニタやDD-8にも逆アセンブラ機能がありますが，これらは1命令ごとにニーモニックに変換するだけです．Daisyは，未知のマシン語プログラムからエディタにかけることのできるザイログニーモニックのソース・プログラムを作成することができます．つまり，アセンブラの全く逆の仕事を行い，ソース・ファイルを作りますから，それをエディタで再編集し，アセンブラ，ローダを通すことによって他機種のマシン語プログラムの移植やアドレスの変更などを簡単に行うことができます．

また，メモリ・ダンプ，アスキー・ダンプ，およびクロスリファレンス・リストを出力することができ，プログラム解析に非常に役立ちます．

逆アセンブルを行う場合に困るのは，そのマシン語プログラムのデータ領域がどこにあるのかわからないことですが，Daisyでは擬似命令として指定することも可能であり，適切な逆アセンブル・リストを得ることができます．

DUADは，5インチまたは8インチのディスケットによって供給され，150ページ強の詳しいマニュアルが付属しています．

7.3　多言語システム…CP/Mについて

最近のパーソナル・コンピュータの広告をみますと，必ずといってよい程CP／Mという文字が出てきます．ではこのCP／Mとは，一体何であるのかを説明しましょう．

CP／Mは，米国デジタルリサーチ社の開発したプログラムの登録商標で，Control Program for Microcomputerの略です．

このプログラムは，ディスクベースの80系マイクロコンピュータシステム上で走らすことのできるオペレーティング・システムです．オペレーティング・システムとは，ユーザがプログラムを作成しやすいように，コンピュータに装備されたハードウェアの周辺装置を制御したり，ファイルの管理などを行うプログラムであり，ユーザに対してある一定の開発環境を与えるものです．

CP／Mもプログラムの一種ですから，他と同様にメモリ上にロードして起動します．そして，それ以降はメモリ上に常駐して，いろいろなアプリケーション・プログラムの実行を管理します．

CP／Mが動作している環境においては，BASICやアセンブラはもちろん，他のいろいろな言語，例えば，FORTRAN, COBOL, PASCAL, FORTH, APL, C, LISPなど，ほとんどの言語処理プログラムがCP／M上で使うことができます．

BASIC,	FORTRAN,	PASCAL
C	, Ada	, COBOL
FORTH ,	LISP	, APL
MUMPS,	PL／I	, etc…

表7.3　CP／M上で使うことのできる高級言語

BASICを例にとると，マイクロソフト社のBASIC-80(MBASIC)，BASICコンパイラ(BASCOM-)，デジタルリサーチ社のCBASIC, Micro Mikes社のbazicというようにそれぞれ特徴をもった数種類のプログラムが各社から発売されていますから，目的に応じて選ぶことができます．

CP／Mのもう1つの利点は，CP／M上で開発されたプログラムは，そこにCP／Mが動いていればたとえ他機種であっても走らせることができるということです．つまり，シャープのMZ-80BのCP／Mで開発したCP／M上で動くプログラムは，メディア変換を行えばPC-8801のCP／M上で使うことができます(もちろん機種のハードウェアに依存するものはだめですが)．これが，先程述べた“ある一定の開発環境を与える”ということでもあり，また多くのソフトウェア・メーカがCP／M上で動くプログラムを作成，販売する理由です．

APPENDIX

A.1　MF-ASM

A.1.1　入力の仕方

MF-ASMは，すべてマシン語で書かれています．リスト1がMF-ASMのマシン語のダンプ・リストです．このダンプ・リストは全体で約5Kバイトありますので，入力の注意事項をよく読んでまちがいのないよう入力してください．

それでは，ダンプ・リストを入力する場合について説明します．

リスト1はMF-ASMのダンプ・リストを入力すると同時に，リストと同じチェックサムを出力するものです．

RUNを実行するとstart addressと表示されますので，B900を入力します．次にアドレスとコロン(:)が表示され，データの入力待ちになります．データの入力は，**リスト2**の通りに16進コードとスペースを順に16バイト分入力した後にRETURNを押します．すると，その行の右端にチェックサムが表示されますので，リストと同じチェックサムが表示されているかを確認してください．

もし違っていた場合には，一度このプログラムを終了して再度起動してください．誤って入力した行のスタート・アドレスを入力し，データの入力を続けます．データ入力時に何も入力せずにRETURNを押すとプログラムは終了します．

このプログラムで入力していったデータは，実際のメモリ・アドレスに格納されますので，1行入力するごとに必ずリストのチェックサムとの確認を行うようにしてください．マシン語のプログラムでは，1バイト入力誤りがあっただけでも正常な動作をしませんので充分注意が必要です．

リスト1

```
100 '
110 '   set meory with check sum for MF-ASM
120 '
130 CLEAR ,&HB8FF : DEFINT A-Z : DIM BYT(15)
140 INPUT "start address";ADR$
150 STADR=VAL("&h"+LEFT$(RIGHT$("000"+ADR$,4),3)+"0")
```

```
160 PRINT RIGHT$("000"+HEX$(STADR),4)+" : "; : LINE INPUT DAT$
170 IF DAT$="" THEN END
180 COUNT=0 : I=1
190 IF I<=LEN(DAT$) AND COUNT<16 GOTO 200 ELSE 280
200   IF MID$(DAT$,I,1)=" " THEN 260
210     BYT(COUNT)=VAL("&H"+MID$(DAT$,I,2))
220     COUNT=COUNT+1
230      IF MID$(DAT$,I,1)<>" " AND I<=LEN(DAT$) GOTO 240 ELSE 260
240       I=I+1
250      GOTO 230
260     I=I+1
270  GOTO 190
280 SUM=0
290 FOR I=0 TO COUNT-1
300   POKE STADR+I,BYT(I) : SUM=(SUM+BYT(I)) MOD 256
310 NEXT
320 LOCATE 60,CSRLIN-1 : PRINT "SUM : ";RIGHT$("0"+HEX$(SUM),2)
330 STADR=STADR+16 : GOTO 160
```

入力が終わったら，とりあえずセーブしておきます．

カセットの場合にはモニタで，

　　h] Wファイル・ネーム, B900, CDFF

ディスクの場合はBASICで

　　BSAVE　〝ファイル・ネーム〞, &HB900, &H1500

を実行します．

リスト2

```
B900 : 00 00 00 CD 98 BA 30 07 3E 0E D3 53 CD C8 BA FF : 16
B910 : 00 00 00 CD CA B9 CD 22 CC 50 41 53 53 2D 31 0D : AD
B920 : 0A 00 3E FF CD D3 BB 00 00 00 CD 22 CC 50 41 53 : 41
B930 : 53 2D 32 0D 0A 00 AF CD D3 BB 3A 57 BF CB 7F C4 : 31
B940 : 52 BD 3A 57 BF CB 67 C2 30 F3 AF 32 5A BF 3E 80 : 2E
B950 : 32 59 BF CD 30 CC CD 22 CC 4F 50 54 49 4F 4E 20 : C7
B960 : 69 73 20 22 20 6C 20 70 20 73 20 6F 20 65 20 22 : 23
B970 : 0D 0A 4F 70 74 69 6F 6E 20 3F 00 CD D9 35 CD 92 : 29
B980 : 5F DA A5 B9 0E 00 23 7E B7 28 08 23 CD A6 B9 28 : A4
B990 : F6 18 B7 79 32 57 BF E6 05 20 8C 79 B7 28 88 FE : FB
B9A0 : 02 28 84 18 98 C9 CB AF FE 4C 20 03 CB C1 C9 FE : 61
B9B0 : 50 20 03 CB C9 C9 FE 45 20 03 CB D1 C9 FE 53 20 : 0C
B9C0 : 03 CB F9 C9 FE 4F C0 CB E1 C9 AF 32 57 BF CD 83 : 59
B9D0 : BB 21 00 C0 11 01 C0 01 FF 3F 36 00 ED B0 21 00 : A1
B9E0 : D0 22 00 C0 CD 87 BB 21 00 C0 11 01 C0 36 00 01 : AB
B9F0 : FF 3F ED B0 C3 C3 BB 22 8D BE CD 83 BB 22 00 C0 : 76
BA00 : C3 C3 BB 3A 92 BE B7 28 04 AF 32 9B BE ED 5B 8D : BD
BA10 : BE ED 53 F1 BE FD 21 9B BE 21 9B BE ED 5B 8D BE : 31
BA20 : AF 47 4E B1 28 2F 23 3A 8F BE B7 28 03 EB 18 1F : FA
BA30 : C5 D5 E5 2A 60 BF 09 CB 7C CA 8F BA E1 ED 5B 60 : B4
BA40 : BF CD 83 BB ED B0 ED 53 60 BF CD C3 BB E1 C1 09 : BC
BA50 : EB ED 53 8D BE 3A 8F BE B7 C2 F2 BB CD CE 35 28 : 1B
BA60 : 28 FE 03 20 05 AF 32 57 BF C9 FE 13 20 1B 3A 57 : EB
BA70 : BF 4F CD 83 35 CB AF FE 50 20 04 CB C9 18 06 FE : 2F
BA80 : 51 20 06 CB 89 79 32 57 BF CD 24 CA C3 F2 BB F1 : A8
```

```
BA90 : F1 F1 CD C3 BB C3 09 C7 21 00 00 39 11 00 B9 ED : D1
BAA0 : 52 D8 CD 22 CC 59 4F 55 20 4D 55 53 54 20 45 58 : 08
BAB0 : 45 43 55 54 45 0D 0A 43 4C 45 41 52 20 2C 26 48 : AE
BAC0 : 42 38 46 46 0D 0A 00 C9 21 33 F3 22 B7 EE 21 75 : 8A
BAD0 : CD 11 30 F3 01 7E 00 ED B0 21 A5 CC 11 00 80 01 : 41
BAE0 : D0 00 CD F0 BA ED B0 CD 00 80 3A 5F BF D3 70 C9 : 95
BAF0 : DB 70 32 5F BF 3E 80 D3 70 C9 CD FB C6 CA 27 C7 : AB
BB00 : CD F6 C6 CA 27 C7 3A 8F BE B7 28 17 CD 3D BB 20 : A3
BB10 : 24 01 08 00 EB 21 87 BE ED B0 1B 1B ED 53 EF BE : 3E
BB20 : C3 C3 BB CD 3D BB CB 7E C2 37 BB 23 CB 7E CD C3 : FF
BB30 : BB C8 C3 3C C7 CB FE CD C3 BB C3 48 C7 CD 87 BB : 3E
BB40 : 21 87 BE 06 06 11 00 00 7B 86 5F 7A CE 00 57 23 : A5
BB50 : 10 F6 62 6B 29 19 7C F6 06 67 29 29 29 29 29 7E : 3F
BB60 : B7 C8 E5 EB 21 87 BE 06 06 1A E6 7F BE 23 13 20 : 54
BB70 : 05 10 F6 E1 05 C9 E1 01 08 00 09 7C B7 20 E0 26 : 06
BB80 : C0 18 DC F3 D3 5C C9 F3 D3 5D C9 CD 3D BB 20 25 : 95
BB90 : 3A 8F BE B7 28 19 3A 90 BE FE 92 20 10 3A EE BE : AD
BBA0 : B7 28 0A 2A EF BE 2B 2B 2B 2B 2B CB FE AF 3D ED : 39
BBB0 : 5B 8D BE 18 0E 23 7E 2F E6 80 28 F3 11 05 00 19 : 4C
BBC0 : 5E 23 56 D3 5F FB C9 CD 87 BB 2A EF BE 73 23 72 : BB
BBD0 : C3 C3 BB 32 8F BE AF 32 62 BF 32 90 BE 21 04 C0 : 27
BBE0 : 22 60 BF 21 00 D0 22 8D BE 22 F1 BE 2A 58 E6 22 : FA
BBF0 : 32 BE 2A 32 BE 22 30 BE 21 90 BE 7E FE 94 CA E1 : 44
BC00 : BC 06 10 CD DE C7 AF 32 EE BE 2A 30 BE CD A4 44 : 9E
BC10 : 7E 23 B6 CA E1 BC CD 5C C7 21 36 BE CD 9D C7 CD : C1
BC20 : 3C BD C3 55 BA FE 3F DC 27 C7 FE 5B D4 27 C7 CD : BA
BC30 : B1 C7 FE 3A 20 1A 22 EB BE CD FA BA 3E FF 32 EE : 93
BC40 : BE CD 4A BD CD 9D C7 CD 3F BD C3 0D BA CD B1 C7 : 5B
BC50 : B7 28 05 FE 20 C4 27 C7 22 EB BE CD E4 C7 21 90 : A8
BC60 : BE FD 21 9B BE 7E CB 7F 20 32 FD 34 00 23 5E FD : FE
BC70 : 73 01 E6 40 C4 24 C1 CD 9A C7 CD 3F BD C3 0D BA : C4
BC80 : CD 34 C7 3A 90 BE FE B0 20 0A FD 34 00 FD 36 01 : 8D
BC90 : C9 C3 0D BA FE 94 C4 27 C7 C3 0D BA DD 21 93 BE : 70
BCA0 : DD 36 00 00 CD 9A C7 CD 3F BD C3 83 BC 3A 90 BE : 94
BCB0 : E6 1F FE 15 D2 2F BD DD 34 00 CD 96 C5 B7 28 11 : FF
BCC0 : CD 4A BD DD 34 00 DD 21 97 BE CD 96 C5 B7 C4 27 : 02
BCD0 : C7 DD 21 93 BE DD 46 01 DD 4E 05 CD 2A BD C3 0D : EE
BCE0 : BA CD 83 BB ED 5B 00 C0 2A 8D BE E5 B7 ED 52 22 : 3F
BCF0 : 02 C0 CD C3 BB E1 4C CD 1B BD 4D CD 1B BD CD 25 : C3
BD00 : CC 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 : AC
BD10 : 20 20 2A 45 4E 44 2A 0D 0A 00 C9 79 CD 3B CC CD : 65
BD20 : 0D 3E 79 CD 3F CC CD 0D 3E C9 3A 90 BE E6 1F 87 : 91
BD30 : 21 F4 C9 5F 16 00 19 5E 23 56 EB E9 FE 2A C8 B7 : BE
BD40 : C8 FE 3B C8 E3 23 23 23 E3 C9 2A EB BE 23 22 EB : C4
BD50 : BE C9 DD 21 57 BF 3A 59 BF CB 7F C4 98 CB 21 00 : 7F
BD60 : C0 22 66 BF CD 20 BE CD CE 35 28 0A FE 03 CA 07 : 86
BD70 : BE FE 13 CC 83 35 2A 66 BF CD AC BD CD C3 BB D2 : F5
BD80 : 07 BE 22 66 BF 54 5D CD A3 BD D2 C0 BD E5 D5 7E : 71
BD90 : E6 7F 47 1A E6 7F B8 20 04 23 13 18 F2 D1 E1 38 : 31
BDA0 : E6 18 E2 01 08 00 18 03 01 06 00 09 CD 87 BB 7C : 9F
BDB0 : B7 C8 7E B7 28 ED 23 23 CB 7E 20 EC 2B 2B 37 C9 : BA
BDC0 : CD C9 BD DA 64 BD C3 67 BD CD DA BD 3A 63 BF 3D : 32
BDD0 : 32 63 BF B7 C0 CD 5C CB 37 C9 CD 87 BB 62 6B 23 : BE
BDE0 : 23 CB FE EB ED 5B 64 BF 06 06 7E E6 7F 12 23 13 : 79
```

```
BDF0 : 10 F8 13 D5 5E 23 56 E1 CD C3 BB CD F7 CB 23 23 : C8
BE00 : 23 23 22 64 BF EB C9 CD C3 BB CD 5C CB 21 02 C0 : 61
BE10 : 11 08 00 CD 87 BB CB BE 19 7C B7 20 F9 C3 C3 BB : 57
BE20 : CD C3 BB CD B0 CA ED 53 64 BF 3E 05 32 63 BF C9 : 55
BE30 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BE40 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BE50 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BE60 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BE70 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BE80 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BE90 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BEA0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BEB0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BEC0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BED0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BEE0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BEF0 : 20 20 20 4D 46 2D 41 53 4D 00 00 00 00 00 00 00 : 01
BF00 : 28 43 29 20 31 39 38 33 20 62 79 20 20 20 20 20 : 24
BF10 : 20 4D 41 43 48 49 4E 45 20 46 52 49 45 4E 44 00 : ED
BF20 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BF30 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BF40 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
BF50 : 00 00 00 00 00 00 00 00 00 80 00 00 00 00 00 00 : 80
BF60 : 00 00 00 00 00 00 00 00 CD F2 C0 CB 7F CA 73 C0 : C6
BF70 : CB 7D CA 0A C0 CB 70 CA C8 BF CB 71 CA AB BF CB : A3
BF80 : 59 20 20 CB 58 20 0A 1E 40 78 CD 19 C1 79 C3 12 : B1
BF90 : C1 CD E3 C0 1E 47 78 E6 04 28 02 CB DB FD 73 01 : 39
BFA0 : C3 24 C1 CD E6 C0 1E 57 79 18 EC CB 6D CA 30 C7 : 06
BFB0 : CD 08 C1 79 CD C9 C0 28 06 1E 0A 50 C3 DC BF 1E : 87
BFC0 : 46 78 CD 19 C1 C3 B0 C1 CB 6D C2 30 C7 CB 6C 1E : DF
BFD0 : F9 28 28 78 CD C9 C0 28 11 1E 02 51 FE 80 28 02 : 69
BFE0 : CB E3 FD 73 01 7A CD E7 C0 C9 1E 70 CD 05 C1 CD : C4
BFF0 : 12 C1 CD A4 C0 CD D6 C0 C3 2C C1 78 FE 8B C2 30 : 0A
C000 : C7 CD B3 C0 FD 73 01 C3 2F C1 CB 70 28 16 CB 6D : DC
C010 : 20 0B 1E 06 CD 08 C1 CD 19 C1 C3 D1 C0 CD E6 C0 : 53
C020 : 1E 3A 18 37 CB 6C 28 1F CB 6D C2 30 C7 CD B0 C0 : 53
C030 : 1E 36 FE 82 20 06 CD D1 C0 C3 2C C1 DD 7E 06 DD : 46
C040 : 77 03 FD 34 00 18 AE 78 CD C5 C0 CB 6D 20 13 1E : C4
C050 : 01 CD 57 C0 C3 2C C1 78 CD 16 C1 FD 34 00 CD D1 : 80
C060 : C0 C9 78 1E 2A E6 C7 FE 82 28 E6 1E 4B CD 57 C0 : D1
C070 : C3 24 C1 E6 20 CA 30 C7 7D FE 80 C2 30 C7 79 FE : 9A
C080 : C7 1E 32 28 18 CD C5 C0 1E 43 38 08 1E 22 CD 9A : F1
C090 : C0 C3 2F C1 CD 9A C0 C3 24 C1 CD 16 C1 FD 34 00 : 17
C0A0 : CD D6 C0 C9 78 18 01 79 FE 82 20 08 FD 35 00 C9 : D9
C0B0 : 78 18 01 79 FE 82 C8 FE 92 C8 FE A2 C8 18 2B FE : 53
C0C0 : 83 20 06 37 C9 FE 8B 28 FA B7 28 1E FE 82 D8 18 : C1
C0D0 : E5 2A 99 BE 18 03 2A 95 BE 22 9D BE 21 9B BE 34 : 29
C0E0 : 23 73 C9 79 18 01 78 FE C7 C8 F1 C3 30 C7 16 01 : B8
C0F0 : 18 02 16 02 FD 34 00 3A 93 BE 67 E6 03 BA 7C DD : 51
C100 : 6E 04 C8 18 E5 51 18 01 50 7A E6 F8 FE C0 7A C8 : 49
C110 : 18 D8 E6 07 18 08 E6 03 07 E6 07 07 07 07 B3 5F : 01
C120 : 32 9C BE C9 3E ED 18 15 3E CB 18 11 78 18 01 79 : E9
C130 : E6 30 C8 E6 20 20 04 3E DD 18 02 3E FD C5 01 03 : 41
C140 : 00 21 9E BE 11 9F BE ED B8 34 23 77 C1 C9 CD F2 : A7
```

```
C150 : C0 CB 6D C2 30 C7 E6 20 1E E3 C2 FB BF 79 FE 82 : 2D
C160 : 20 07 78 FE 81 1E EB 18 08 FE 83 C2 30 C7 B8 1E : 57
C170 : 08 C2 30 C7 FD 73 01 C9 CD EE C0 CD 1B C2 FE 81 : 9F
C180 : C2 30 C7 78 CD BF C0 18 4F CD F2 C0 E6 20 C2 30 : 5B
C190 : C7 CD 1B C2 78 FE C7 20 20 CB 6D 20 13 CB 5D 20 : A1
C1A0 : 0A 7B E6 F8 5F CD 05 C1 C3 12 C1 CB F3 C3 D1 C0 : FD
C1B0 : CD A7 C0 CD D1 C0 C3 2F C1 CB 6D C2 30 C7 CD B0 : B3
C1C0 : C0 79 CD C5 C0 7B FE 86 79 20 13 1E 09 E6 C7 FE : 08
C1D0 : 82 79 20 04 B8 C2 30 C7 CD 16 C1 C3 2C C1 E6 30 : FA
C1E0 : C2 30 C7 78 FE 82 C2 30 C7 7B FE 8E 1E 4A 28 02 : 03
C1F0 : 1E 42 79 CD 16 C1 C3 24 C1 CD EE C0 CD 1B C2 28 : 72
C200 : 10 E6 20 C2 F2 BF 7B E6 F8 5F CD 08 C1 CD 12 C1 : 77
C210 : C9 CB 6C C2 30 C7 CB F3 C3 D6 C0 3A 91 BE 5F 7C : 34
C220 : CB 7C C9 CD EE C0 CD 1B C2 CA 30 C7 E6 20 C2 F2 : B0
C230 : BF 78 CB 77 20 10 CD C5 C0 7B 1E 0B FE 34 20 02 : F3
C240 : 1E 03 78 C3 D8 C1 CD 0A C1 7B EE 30 5F 78 C3 19 : D9
C250 : C1 CD EE C0 FE 09 C2 30 C7 CD 1B C2 3A 95 BE FE : 31
C260 : 03 D2 30 C7 FE 01 38 06 CB E3 28 02 CB DB FD 73 : F7
C270 : 01 C3 24 C1 CD EE C0 FE 09 C2 30 C7 CD 1B C2 3A : C8
C280 : 95 BE 57 A3 C2 30 C7 7A C3 1E C1 CD EE C0 CD 1B : 85
C290 : C2 CA 30 C7 E6 20 28 16 CD B0 C0 FD 73 01 FE 82 : F5
C2A0 : 28 06 CD D6 C0 CD BB C2 CD 28 C1 C3 2C C1 7B E6 : A2
C2B0 : F8 5F CD 08 C1 CD 12 C1 C3 28 C1 23 7E 73 2B 77 : EF
C2C0 : C9 CD F2 C0 FE 0A C2 30 C7 CD 1B C2 3A 95 BE FE : 3E
C2D0 : 08 D2 30 C7 CD 19 C1 CB 6D 28 16 CD B3 C0 FD 73 : 9E
C2E0 : 01 FE 82 28 06 CD D1 C0 CD BB C2 CD 28 C1 C3 2F : FF
C2F0 : C1 7B E6 F8 5F CD 05 C1 18 BB CD F2 C0 FE 82 C2 : A0
C300 : 30 C7 7D FE 28 28 16 FE A0 C2 30 C7 79 FE C1 C2 : 29
C310 : 30 C7 1E 40 CD 08 C1 CD 19 C1 C3 24 C1 CD E6 C0 : AD
C320 : CD 1B C2 C3 D1 C0 CD F2 C0 FE 2A 28 12 FE A2 C2 : 41
C330 : 30 C7 78 FE C1 C2 30 C7 1E 41 CD 05 C1 18 D8 CD : 96
C340 : E3 C0 18 38 FD 34 00 3A 93 BE FE 09 C9 CD 44 C3 : 53
C350 : 28 27 FE A1 28 19 1E C2 FE 12 C2 30 C7 3A 97 BE : 67
C360 : FE 08 C2 30 C7 78 CD 19 C1 FD 34 00 C3 D1 C0 CD : 30
C370 : B0 C0 FD 36 01 E9 C3 2C C1 FD 34 00 CD 1B C2 C3 : DB
C380 : D6 C0 CD 44 C3 28 2A FE 12 C2 30 C7 3A 97 BE FE : 12
C390 : 08 C2 30 C7 78 FE 04 D2 30 C7 F6 04 1E 00 CD 19 : 02
C3A0 : C1 32 91 BE 2A 99 BE 18 0B CD EE C0 FE 09 C2 30 : 5A
C3B0 : C7 2A 95 BE ED 5B 8D BE 13 13 B7 ED 52 CD 1B C2 : 9D
C3C0 : 38 0D B7 C2 23 C7 7D FE 80 D2 23 C7 C3 D9 C0 3C : F7
C3D0 : C2 23 C7 7D FE 80 DA 23 C7 C3 D9 C0 CD 44 C3 1E : B9
C3E0 : C4 C2 58 C3 C3 79 C3 CD EE C0 FE 11 C2 30 C7 1E : 01
C3F0 : C0 78 C3 19 C1 CD 47 C3 C2 30 C7 3A EE BE B7 CA : CC
C400 : 38 C7 3A 92 BE E6 01 C0 3A 8F BE B7 ED 5B 95 BE : 09
C410 : C2 C7 BB F1 C3 11 BA CD 47 C3 C2 30 C7 3A 62 BF : AE
C420 : B7 2A 95 BE ED 5B 8D BE 20 0F 3D 32 62 BF E5 21 : 8C
C430 : 00 D0 ED 52 E1 D2 F7 B9 B7 ED 52 DA 30 C7 EB 19 : 3D
C440 : 22 8D BE 2A 60 BF 19 22 60 BF C9 C3 30 C7 47 CD : A7
C450 : 47 C3 C8 C3 27 C7 2A 8D BE 22 F1 BE 21 00 00 22 : 0C
C460 : 5D BF CD 8B C5 CD 4E C4 ED 5B 95 BE 2A 5D BF 19 : 12
C470 : 22 5D BF 78 B7 CA 80 C4 CD D9 C7 CD 4A BD 18 E2 : B6
C480 : 3A 92 BE B7 C2 15 BA EB 2A 8D BE 19 DC 1F C7 22 : 2F
C490 : 8D BE 2A 60 BF 19 22 60 BF C3 15 BA 3E 01 32 B7 : A8
C4A0 : C4 3C 32 BA C4 CD 8B C5 CD 4E C4 FD 34 00 FD 34 : 0E
```

```
C4B0 : 00 ED 5B 95 BE FD 73 01 FD 72 02 78 B7 CA 03 BA : 33
C4C0 : CD D9 C7 CD 4A BD 3A BA C4 3C 18 D2 3E 01 32 F4 : 84
C4D0 : C4 CD 8B C5 CD 4E C4 3A 95 BE CD EF C4 78 B7 CA : C6
C4E0 : 03 BA 3E 97 32 90 BE CD D9 C7 CD 4A BD 18 E2 FD : 4A
C4F0 : 34 00 FD 77 01 E5 21 F4 C4 34 E1 C9 F1 CD 50 C5 : 18
C500 : 28 06 4F CD EF C4 18 F5 B7 CC 27 C7 CD 4D BD AF : 01
C510 : 32 90 BE DD 71 02 CD 05 C6 CD 4E C4 3A 95 BE E5 : B9
C520 : 21 F4 C4 35 CD F2 C4 E1 18 B3 DD CB 00 DE 3A 90 : 8D
C530 : BE FE 97 28 C7 11 00 00 CD 50 C5 28 0D 5F CD 50 : E6
C540 : C5 28 07 53 5F CD 50 C5 20 FB B7 CA 27 C7 18 13 : 3D
C550 : 23 7E FE 27 28 02 B7 C9 23 7E FE 27 28 F8 2B 7E : FF
C560 : FE 27 C9 22 EB BE CD 36 C6 CD 4A BD C3 05 C6 06 : EA
C570 : 00 FE 2B 28 04 FE 2D C0 05 F1 78 23 32 ED BE DD : 8B
C580 : CB 00 DE CD BF C5 C2 E0 C5 18 23 DD 21 93 BE FD : E8
C590 : 21 9B BE DD 34 00 AF 32 ED BE CD 9A C7 FE 28 20 : 8B
C5A0 : 05 DD CB 00 EE 23 CD BF C5 20 22 CD 6F C5 FE 27 : 77
C5B0 : CA 2A C5 FE 24 ED 5B 8D BE CA 66 C5 C3 27 C7 CD : E1
C5C0 : B1 C7 22 EB BE 47 3A 87 BE FE 20 78 C9 3A 90 BE : F0
C5D0 : CB 6F 28 06 CD E3 C6 CA 63 C6 CD ED C6 CA 6F C6 : 50
C5E0 : CD ED C6 CA 34 C7 CD E3 C6 CA 34 C7 3A 87 BE FE : FD
C5F0 : 3F 30 09 CD 53 C6 B7 CA 27 C7 18 0D CD 8B BB CC : D1
C600 : 3C C7 CD 36 C6 DD CB 00 DE CD 9A C7 FE 2E 20 12 : DE
C610 : 23 CD B4 C7 CD 9D C7 CD D4 C6 DA 27 C7 CD C8 C6 : 26
C620 : 18 E7 FE 29 28 05 CD 6F C5 18 38 DD CB 00 6E CA : 84
C630 : 27 C7 23 C3 96 C6 3A ED BE B7 28 09 AF 32 ED BE : 89
C640 : 67 6F ED 52 EB DD 66 03 DD 6E 02 19 DD 74 03 DD : DD
C650 : 75 02 C9 DD CB 00 DE CD 4A CC B7 C8 EB CD 36 C6 : DC
C660 : 3E FF C9 DD CB 00 6E C2 27 C7 2A EB BE 18 27 2A : 08
C670 : EB BE DD CB 00 FE DD CB 00 6E 28 1A DD 7E 01 FE : 01
C680 : C1 28 0A CB 77 C2 27 C7 E6 30 C2 09 C6 CD 9D C7 : BD
C690 : FE 29 C2 27 C7 23 DD 7E 00 E6 A0 FE A0 20 1B DD : 91
C6A0 : 7E 03 B7 CA B3 C6 3C C2 1F C7 DD CB 02 7E CA 1F : 70
C6B0 : C7 18 07 DD CB 02 7E C2 1F C7 CD 9D C7 B7 C8 FE : 64
C6C0 : 2C C8 D6 3B C8 C3 27 C7 00 00 00 DD 66 02 DD 6E : 0E
C6D0 : 03 C3 4C C6 21 F0 C9 11 87 BE CD 22 C8 D8 7E C8 : DD
C6E0 : 23 18 F4 CD F6 C6 C0 DD CB 00 E6 18 04 CD FB C6 : B0
C6F0 : C0 7E DD 77 01 C9 21 DB C9 18 03 21 B3 C9 11 87 : 71
C700 : BE CD 22 C8 D8 C8 23 18 F5 CD 22 CC 42 55 46 46 : 23
C710 : 45 52 20 4F 56 45 52 20 45 52 52 4F 52 00 C9 06 : 6C
C720 : 04 18 06 06 04 18 23 06 20 CD 4A C7 F1 C3 55 BA : 2E
C730 : 06 10 18 16 06 10 18 F1 06 20 18 0E 06 02 18 0A : D9
C740 : 06 02 18 E5 06 08 18 E1 06 01 2A 8D BE 22 F1 BE : 59
C750 : 3A 92 BE B0 32 92 BE AF 32 9B BE C9 2A 30 BE CD : A4
C760 : A4 44 11 32 BE 01 04 00 ED B0 7E FE 8F 20 09 23 : E2
C770 : 7E FE 20 28 16 C3 AB C7 FE 3A C2 AB C7 23 7E FE : 1A
C780 : 8F C2 AB C7 23 7E FE E9 C2 AB C7 23 01 50 00 7E : 71
C790 : B7 ED A0 C8 EA 8F C7 AF 12 C9 2A EB BE 22 EB BE : 74
C7A0 : 7E FE 09 28 03 FE 20 C0 23 18 F2 00 00 F1 C3 F2 : 61
C7B0 : BB CD 9D C7 E5 06 06 21 87 BE E5 3E 20 CD DF C7 : F9
C7C0 : D1 E1 06 06 7E FE 30 D8 FE 3A 38 06 FE 3F D8 FE : CB
C7D0 : 5B D0 12 13 23 10 ED 7E C9 06 04 21 93 BE AF 77 : 59
C7E0 : 23 10 FC C9 3A 87 BE FE 59 D2 44 C7 D6 41 DA 44 : E0
C7F0 : C7 87 26 00 6F 11 42 C8 19 5E 23 56 23 4E 23 46 : C8
C800 : EB 11 88 BE C5 CD 22 C8 C1 DA 44 C7 20 08 5E 23 : 0D
```

```
C810 : 56 ED 53 90 BE C9 23 23 E5 ED 42 E1 DA 01 C8 C3 : 4E
C820 : 44 C7 1A 4F 7E B7 28 0F 47 E6 7F B9 23 13 37 3F : F1
C830 : 20 08 CB 78 28 EC 1A D6 20 C9 78 E6 80 C0 7E 23 : 97
C840 : 18 F9 74 C8 80 C8 84 C8 A6 C8 D6 C8 B2 C9 B2 C9 : E3
C850 : E8 C8 ED C8 09 C9 B2 C9 0F C9 B2 C9 24 C9 2C C9 : E7
C860 : 4B C9 B2 C9 54 C9 92 C9 B2 C9 B2 C9 B2 C9 B2 C9 : F3
C870 : AE C9 B2 C9 44 C3 83 8E 44 C4 83 86 4E C4 84 A6 : 57
C880 : 49 D4 89 46 41 4C CC AF CD 43 C6 00 3F D0 84 BE : 1B
C890 : 50 C4 40 A9 50 44 D2 40 B9 50 C9 40 A1 50 49 D2 : C1
C8A0 : 40 B1 50 CC 00 2F 41 C1 00 27 C2 97 00 C3 97 00 : 18
C8B0 : 45 C3 86 35 45 46 C2 97 00 45 46 CD 97 00 45 46 : 21
C8C0 : D7 96 00 45 46 D3 95 00 C9 00 F3 4A 4E DA 8E 10 : 2C
C8D0 : D3 95 00 D7 96 00 C9 00 FB 4E C4 94 00 51 D5 92 : F7
C8E0 : 00 D8 81 E3 58 D8 00 D9 41 4C D4 00 76 CD C7 46 : F6
C8F0 : CE 8A DB 4E C3 85 34 4E C4 40 AA 4E 44 D2 40 BA : 57
C900 : 4E C9 40 A2 4E 49 D2 40 B2 D0 AC C3 D2 AD 18 C4 : EE
C910 : 80 00 44 C4 40 A8 44 44 D2 40 B8 44 C9 40 A0 44 : F3
C920 : 49 D2 40 B0 45 C7 40 44 4F D0 00 00 D2 84 B6 52 : 18
C930 : C7 93 00 54 44 D2 40 BB 54 49 D2 40 B3 55 D4 8B : D5
C940 : D3 55 54 C4 40 AB 55 54 C9 40 A3 4F D0 82 C1 55 : 37
C950 : 53 C8 82 C5 45 D3 89 86 45 D4 B0 C9 45 54 C9 40 : BD
C960 : 4D 45 54 CE 40 45 CC 88 16 4C C1 00 17 4C C3 88 : 5E
C970 : 06 4C 43 C1 00 07 4C C4 40 6F D2 88 1E 52 C1 00 : A7
C980 : 1F 52 C3 88 0E 52 43 C1 00 0F 52 C4 40 67 53 D4 : 13
C990 : 91 C7 42 C3 83 9E 43 C6 00 37 45 D4 89 C6 4C C1 : 33
C9A0 : 88 26 52 C1 88 2E 52 CC 88 3E 55 C2 84 96 4F D2 : AD
C9B0 : 84 AE 00 C1 C7 C2 C0 C4 C2 C5 C3 C8 C4 CC C5 C9 : 30
C9C0 : C8 D2 CC C3 C1 42 C3 80 44 C5 81 48 CC 82 53 D0 : B2
C9D0 : 8B 49 D8 92 49 D9 A2 41 C6 83 00 4E DA 00 DA 01 : 8F
C9E0 : 4E C3 02 C3 03 50 CF 04 50 C5 05 D0 06 CD 07 00 : C0
C9F0 : 52 B8 00 00 68 BF 4E C1 78 C1 89 C1 F9 C1 23 C2 : 62
CA00 : 23 C2 51 C2 8B C2 C1 C2 FA C2 26 C3 4D C3 82 C3 : C2
CA10 : A9 C3 DC C3 E7 C3 74 C2 F5 C3 17 C4 4B C4 56 C4 : A7
CA20 : 9C C4 CC C4 DD 21 57 BF CD B0 CA DD CB 00 46 20 : 59
CA30 : 1B 3A 92 BE B7 C8 DD CB 00 56 28 10 DD CB 00 8E : 90
CA40 : EB CD ED CB 36 27 23 00 EB C3 4D CB DD CB 02 7E : DE
CA50 : C4 98 CB 21 36 BE CD 9D C7 B7 CA 50 CB FE 2A CA : FB
CA60 : 50 CB FE 3B 11 02 BF CA 50 CB 3A 92 BE B7 C2 39 : 47
CA70 : CB 11 9B BE 1A 13 ED 53 5B BF FE 05 32 58 BF 30 : 38
CA80 : 09 CD BD CA CD D6 CA C3 F2 CA CD BD CA CD D4 CA : 08
CA90 : CD F2 CA 3A 58 BF D6 04 32 58 BF FE 05 F5 38 02 : 2F
CAA0 : 3E 04 CD BD CA CD D6 CA EB CD 5C CB F1 D8 18 E3 : A6
CAB0 : 21 F3 BE 54 5D 06 64 3E 20 CD DF C7 C9 F5 3A 90 : 46
CAC0 : BE FE 93 28 0A 21 F3 BE ED 5B F1 BE CD F7 CB 21 : FA
CAD0 : F8 BE F1 C9 3E 04 47 04 05 C8 ED 5B 5B BF 1A 13 : 59
CAE0 : ED 53 5B BF CD FC CB E5 2A F1 BE 23 22 F1 BE E1 : 81
CAF0 : 18 E6 EB 2A EB BE CD B4 C7 FE 3A 2A EB BE 20 0E : 3D
CB00 : 11 02 BF 7E 12 FE 3A 23 13 20 F8 CD 9D C7 11 09 : 33
CB10 : BF 01 53 BF 7E B7 28 44 FE 20 28 0E FE 3B 28 14 : 3C
CB20 : CD 8F CB 28 37 22 EB BE 18 EA 11 0E BF CD 9D C7 : 62
CB30 : FE 3B 20 1C 11 21 BF 18 17 CD BD CA 36 2F 23 CD : 3E
CB40 : FC CB 36 28 23 CD ED CB 36 29 23 18 A5 21 36 BE : 21
CB50 : 01 53 BF 7E B7 28 05 CD 8F CB 20 F7 3E 0D 12 13 : 23
CB60 : 3E 0A 12 13 AF 12 21 F3 BE 7E 23 B7 28 10 FE 0C : 9A
```

```
CB70 : 28 03 CD 0D 3E DD CB 00 4E C4 D4 3E 18 EB DD 35 : 24
CB80 : 02 C0 11 F3 BE 3E 0C 12 CD 63 CB CD 98 CB C9 12 : E6
CB90 : 13 23 78 BA C0 79 BB C9 DD 36 02 3E CD B0 CA CD : 8C
CBA0 : B0 CB EB CD 5C CB 11 F3 BE CD 5C CB CD B0 CA C9 : 20
CBB0 : 21 CA CB 11 F3 BE 01 23 00 ED B0 DD 34 03 DD 5E : 88
CBC0 : 03 16 00 21 39 BF CD F1 CB C9 2A 2A 20 20 20 4D : 85
CBD0 : 46 2D 41 53 53 45 4D 42 4C 45 52 28 31 29 20 28 : DB
CBE0 : 50 43 2D 38 38 30 31 29 20 20 20 2A 2A ED 5B 34 : EA
CBF0 : BE CD 09 CC CD 02 CC 7A CD FC CB 7B 4F CD 3B CC : A7
CC00 : 77 23 79 CD 3F CC 77 23 C9 E5 EB 01 00 10 11 00 : 40
CC10 : 00 29 7B 8F 27 5F 7A 8F 27 57 79 8F 27 4F 10 F1 : BF
CC20 : E1 C9 CD 30 CC E3 7E B7 23 E3 C8 CD 0D 3E 18 F5 : 7E
CC30 : 3E 0D CD 0D 3E 3E 0A CD 0D 3E C9 0F 0F 0F 0F E6 : AE
CC40 : 0F FE 0A 38 02 C6 07 C6 30 C9 11 87 BE 21 8D BE : 9F
CC50 : 06 06 2B 05 7E FE 20 28 F9 04 05 FA A3 CC FE 48 : B1
CC60 : 20 09 3E 29 32 7C CC 0E 07 18 08 3E 09 32 7C CC : 00
CC70 : 04 0E 00 21 00 00 C5 29 4D 44 29 29 09 1A FE 30 : 55
CC80 : 38 20 FE 3A 38 0F C1 0C 0D 28 18 FE 47 30 14 FE : 78
CC90 : 41 38 10 91 C5 D6 30 4F 06 00 09 13 C1 10 D7 3E : 3C
CCA0 : FF C9 F1 AF C9 18 05 18 71 18 2D C9 D9 06 5F 1E : 41
CCB0 : 5E D9 F3 CD CA 80 21 00 B9 22 00 C0 21 F3 14 22 : 47
CCC0 : 02 C0 CD C6 80 FB 21 00 B9 11 04 C0 01 F3 14 F3 : 7A
CCD0 : CD AC 80 CD C6 80 FB C9 CD 96 F3 4C 4F 41 44 20 : 66
CCE0 : 4F 46 46 53 45 54 3F 20 00 CD 92 5F D8 11 00 00 : CD
CCF0 : 23 7E B7 28 20 FE 47 30 DF FE 41 38 06 CB AF D6 : C1
CD00 : 37 18 08 D6 30 38 D1 FE 0A 30 CD EB 29 29 29 29 : FA
CD10 : EB B3 5F 18 DB D9 06 5C 18 06 11 00 00 D9 06 5E : 97
CD20 : 1E 5F D9 F3 CD 98 80 CD AC 80 FB D0 CD 96 F3 4C : 94
CD30 : 4F 41 44 20 45 52 52 4F 52 3F 3F 00 C9 D9 48 ED : D3
CD40 : 79 D9 2A 00 C0 19 EB 21 04 C0 ED 4B 02 C0 D3 5F : 51
CD50 : C9 78 B1 C8 CD C6 80 7E CD CA 80 08 7A FE E6 30 : F8
CD60 : 07 08 12 23 13 0B 18 E9 08 37 C9 D9 48 18 02 D9 : 7F
CD70 : 4B ED 79 D9 C9 18 4E 00 3E 0E D3 53 22 AC F3 CD : B9
CD80 : 78 F3 CD 10 B9 2A AC F3 CD 96 F3 52 45 54 55 52 : B2
CD90 : 4E 20 54 4F 20 42 41 53 49 43 20 4F 52 20 4D 4F : 10
CDA0 : 4E 49 54 4F 52 20 28 42 2F 4D 29 20 3F 00 CD 83 : 6A
CDB0 : 35 CB AF FE 42 C8 FE 4D CA 26 E8 18 C8 CD 8C F3 : 06
CDC0 : CD 02 80 18 06 CD 8C F3 CD 04 80 3A AB F3 D3 70 : 25
CDD0 : C9 DB 70 32 AB F3 3E 80 D3 70 C9 3E 0D CD 0D 3E : 11
CDE0 : 3E 0A CD 0D 3E E3 7E B7 23 E3 C8 CD 0D 3E 18 F5 : 6B
CDF0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
```

A．1．2　MF-ASMのロードとイニシャライズ

MF-ASMを使う際にはまず

```
CLEAR, &HB8FF : SCREEN 0, 2
```

を実行してから，BLOAD命令やモニタのRコマンドでメモリ上にMF-ASM本体をロードします．

その後，モニタに入って

```
h] GB900
```

を実行します．これは，MF-ASMのイニシャライズを行うもので，ロード後必ず一度実行するよ

うにします．MF-ASMの実行に関するアドレスで覚えておかなければならないのは，このB900Hだけです．

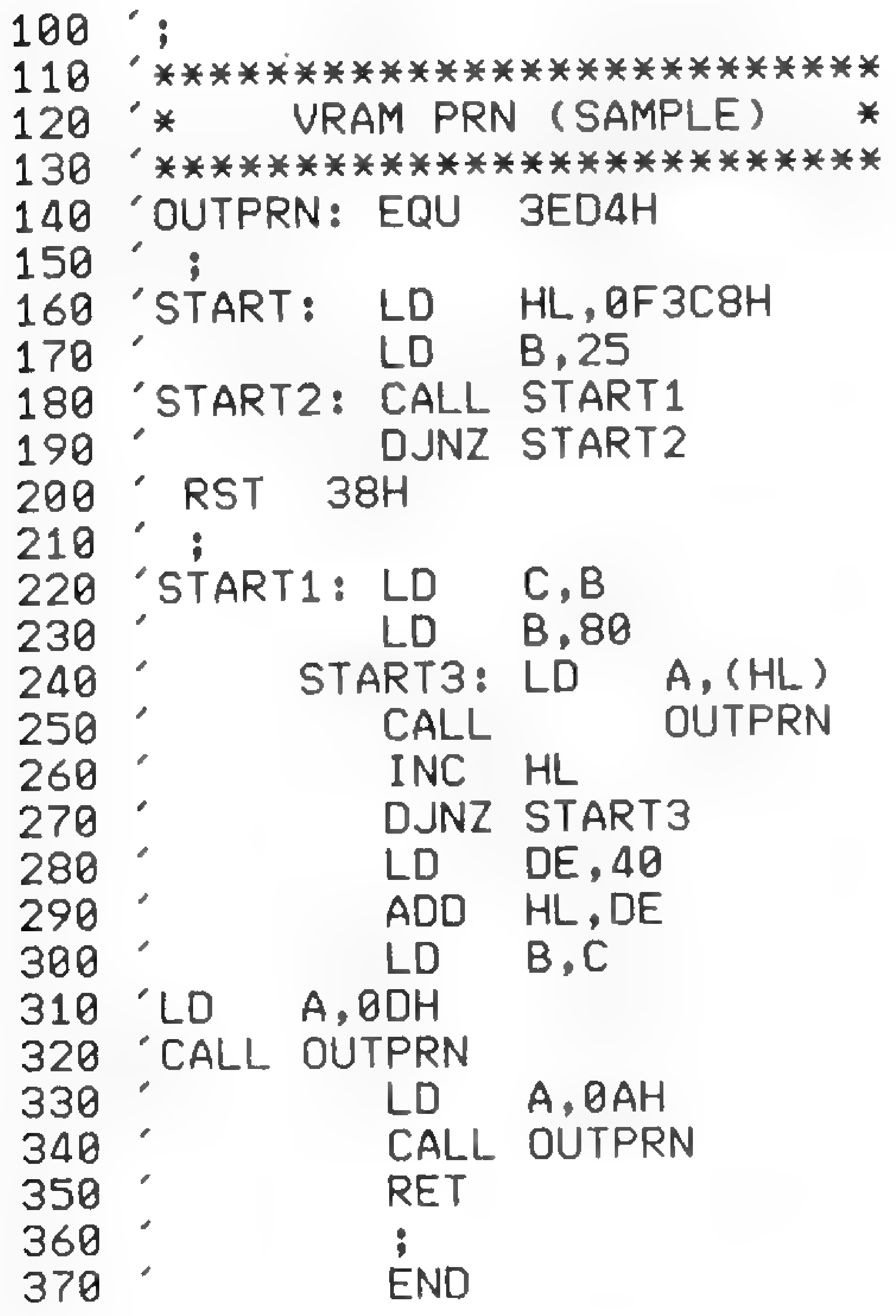
A．1．3　ソース・プログラムの作り方

MF-ASMがアセンブルするソース・プログラムは，BASICの注釈行として作成します(リスト3参照)．注釈行とは，行番号の直後にアポストロフィ(')またはREMがある行のことをいい，注釈行でなければMF-ASMはその行を無視します．

実質のソース・プログラムとなるニーモニックは，この注釈行中に後で述べる文法に従って入力していきます．特に大文字を使用しなければならないことに注意してください．

アセンブル・リスト出力において，ある程度書式を整えますから不要なスペースをできるだけ省くようにすれば，テキスト・エリアを有効に使うことができます．つまり，ソース・プログラムの大きさはテキスト・エリアの限界によって決まります．

リスト3



```
100 ';
110 '**************************
120 '*    VRAM PRN (SAMPLE)   *
130 '**************************
140 'OUTPRN: EQU  3ED4H
150 ' ;
160 'START:  LD   HL,0F3C8H
170 '        LD   B,25
180 'START2: CALL START1
190 '        DJNZ START2
200 ' RST  38H
210 ' ;
220 'START1: LD   C,B
230 '        LD   B,80
240 '    START3: LD   A,(HL)
250 '        CALL      OUTPRN
260 '        INC  HL
270 '        DJNZ START3
280 '        LD   DE,40
290 '        ADD  HL,DE
300 '        LD   B,C
310 'LD   A,0DH
320 'CALL OUTPRN
330 '        LD   A,0AH
340 '        CALL OUTPRN
350 '        RET
360 '        ;
370 '        END
```

A．1．4　MF-ASMの起動(アセンブル)

ソース・プログラムの入力が終わり，MF-ASMのロードとイニシャライズが完了すれば，次はいよいよアセンブルです．

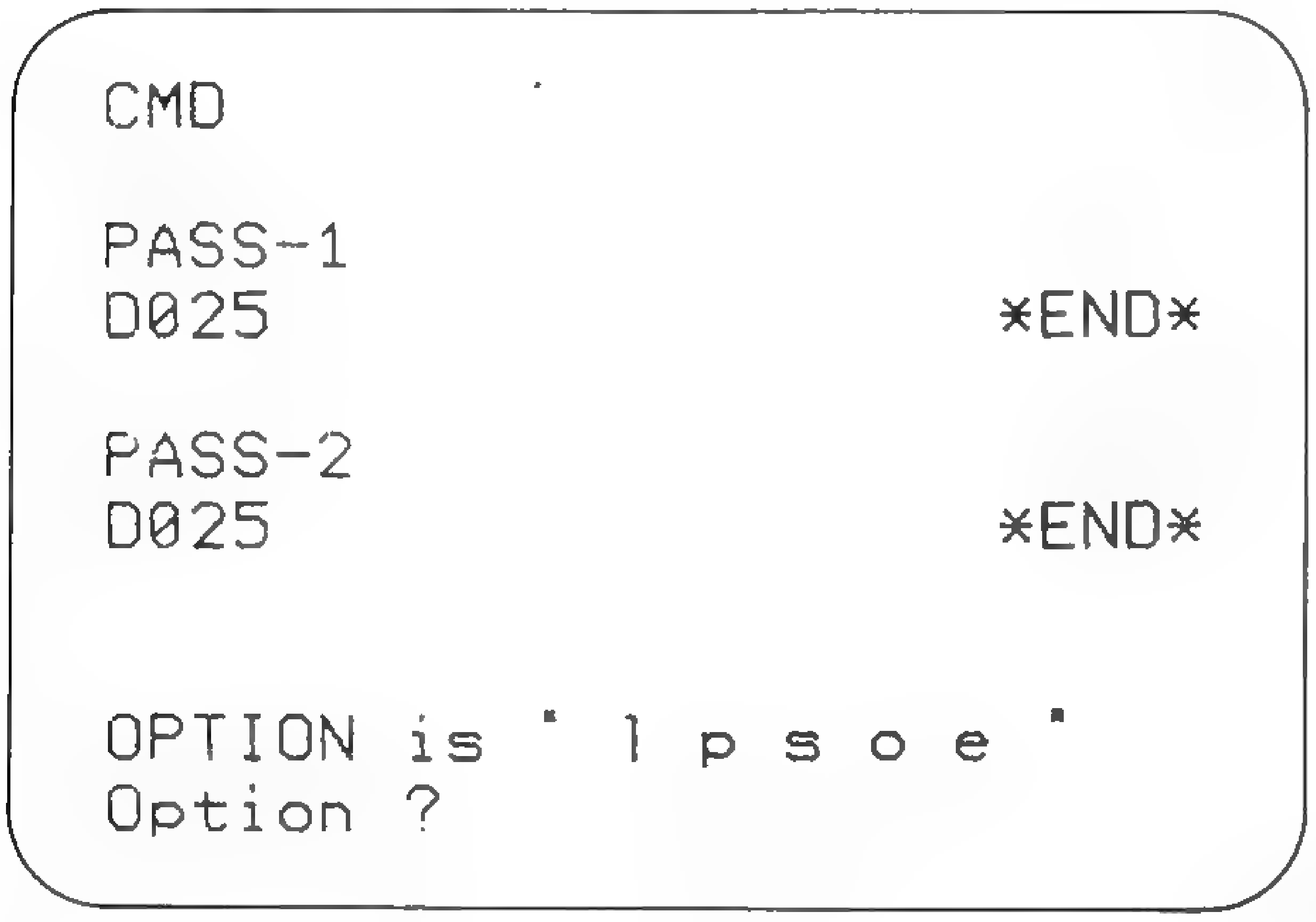
アセンブルは，BASICにおいてCMDを実行することにより行われます．

CMDは，BASICの予約語として登録されていますが，使用されていません．そこで先のイニシャライズにおいて，CMDを実行するとアセンブルを行うようにインストラクションジャンプ・テーブルのCMDの部分を変更しています．

さて図A.1は，リスト3のソース・プログラムを入力した後にCMDを実行した時の画面表示です．

もしエラーがあれば，PASS-2のときにエラーの種類と，その行の表示を行います．D025Hは，生成されるマシン語プログラムの最後のアドレス＋1の値を示しています．

リスト3のソース・プログラムにはORG命令がありません．MF-ASMでは，この場合デフォルトとしてD000Hからのオブジェクトを生成します．なお，オブジェクトはこの時点ではG-VRAM（BLUE）上に出力されており，メイン・メモリ上には存在しません．そのため，オブジェクトのメモリイメージは16Kバイトを超えることはできません．



```
CMD

PASS-1
D025                    *END*

PASS-2
D025                    *END*


OPTION is ' l p s o e '
Option ?
```

図A.1　CMD実行後の表示

A.1.5　オプション・コマンド

ひと通りPASS-1，PASS-2が終了すると，オプション（OPTION）コマンド待ちになります．ここでは，l, p, s, o, eの5つのコマンドを使用できます．それぞれの動作を以下に示します．

l………アセンブルリストを出力する
p………プリンタへの出力スイッチ
s………ラベルのソート出力
o………生成されたオブジェクトをメイン・メモリへロードする．
e………エラー行をソース・プログラムの形式でCRTに出力する

リスト3のプログラムのアセンブルリストをリスト4に，ラベル出力をリスト5に示します．

リスト4

```
**   MF-ASSEMBLER(1) (PC-8801)   **

               ;
**************************
*    VRAM PRN (SAMPLE)   *
**************************
3ED4           OUTPRN:EQU  3ED4H
               ;
D000 21C8F3    START: LD   HL,0F3C8H
D003 0619             LD   B,25
D005 CD0BD0    START2:CALL START1
D008 10FB             DJNZ START2
D00A FF               RST  38H
               ;
D00B 48        START1:LD   C,B
D00C 0650             LD   B,80
D00E 7E        START3:LD   A,(HL)
D00F CDD43E           CALL OUTPRN
D012 23               INC  HL
D013 10F9             DJNZ START3
D015 112800           LD   DE,40
D018 19               ADD  HL,DE
D019 41               LD   B,C
D01A 3E0D             LD   A,0DH
D01C CDD43E           CALL OUTPRN
D01F 3E0A             LD   A,0AH
D021 CDD43E           CALL OUTPRN
D024 C9               RET
               ;
D025                  END
```

リスト5

```
**   MF-ASSEMBLER(1) (PC-8801)   **

               ;
**************************
*    VRAM PRN (SAMPLE)   *
**************************
3ED4           OUTPRN:EQU  3ED4H
               ;
D000 21C8F3    START: LD   HL,0F3C8H
D003 0619             LD   B,25
D005 CD0BD0    START2:CALL START1
D008 10FB             DJNZ START2
D00A FF               RST  38H
               ;
D00B 48        START1:LD   C,B
D00C 0650             LD   B,80
D00E 7E        START3:LD   A,(HL)
D00F CDD43E           CALL OUTPRN
D012 23               INC  HL
D013 10F9             DJNZ START3
D015 112800           LD   DE,40
D018 19               ADD  HL,DE
D019 41               LD   B,C
D01A 3E0D             LD   A,0DH
```

```
D01C CDD43E           CALL OUTPRN
D01F 3E0A             LD   A,0AH
D021 CDD43E           CALL OUTPRN
D024 C9               RET
               ;
D025                  END
OUTPRN 3ED4    START  D000    START1 D00B    START2 D005    START3 D00E
```

プリンタ出力スイッチpはl，sと共に指定した場合CRTに出力したものと同じものをプリンタに出力します．ただ，プリンタへは62行ごとに用紙送りのコードを出力するようになっています．

リストやラベル出力は，CTRL-C(STOP)で停止，CTRL-Sで中断することができます．また，リスト出力時にCTRL-Sで中断している状態では，Pでプリンタ出力，QでCRT出力のみ切り換えを行うことができますから，必要な部分だけリストをとることが可能です．

エラーの行はリストの本来マシン語が書かれる部分に，スラント〝/〟につづいてエラー・コード，そして，カッコで囲ってソース・プログラムの行番号を5桁で表示します．エラー行の命令に対するマシン語は，たとえ何らかのコードが生成されたとしても，正しくないものと考えてください．

エラー・コードの1バイトは，その各ビットがエラーの種類を表しています．

ビット5………文法の誤り
ビット4………オペランドが不適当
ビット3………存在しない命令
ビット2………ディスプレイスメントが不適当
ビット1………ラベルの未定義
ビット0………ラベルの重複定義

oコマンドはオブジェクトをメインRAMにロードするコマンドです．オブジェクトはPASS 2を行うたびにG-VRAM (BLUE) に生成されていますが，メインRAMにはないため，マシン語を実行する前には必ずoコマンドを実行しなければなりません．

oコマンドを実行するとアセンブラは以下のようにオフセットの値を聞いてきます．

```
LOAD OFFSET？
```

オフセットとは，実際のプログラムのスタート・アドレスとの変位値のことをいいます．通常はプログラムにORGが設定されていますので，リターンキーだけを押せばそのアドレスへプログラムが転送されます．ところがマシン語プログラムをメモリに置く場合には，CLEAR文の第2パラメータによる制限があると共に，ワーク・エリアにはプログラムをロードできません．

このようにプログラムの置けないアドレスに割り当てられているプログラムをロードしたい場合には，オフセットが使われます．例えば0000Hから初まるプログラムをロードする場合にはB900H番地より大きいアドレスを指定するオフセットを付けてメモリ上にロードすればプログラムの

スタート・アドレスにかかわらずロードできることになります．ただし，ロード時にプログラムがBASICのワーク・エリアにかかる場合には，エラーが表示されます．

oコマンドが終了した時とコマンド入力時に STOP を押したときは，次のメッセージ表示がなされますので，そのときの必要に応じてBまたはMの入力を行います．

```
RETURN TO BASIC OR MONITOR(B/M) ?
```

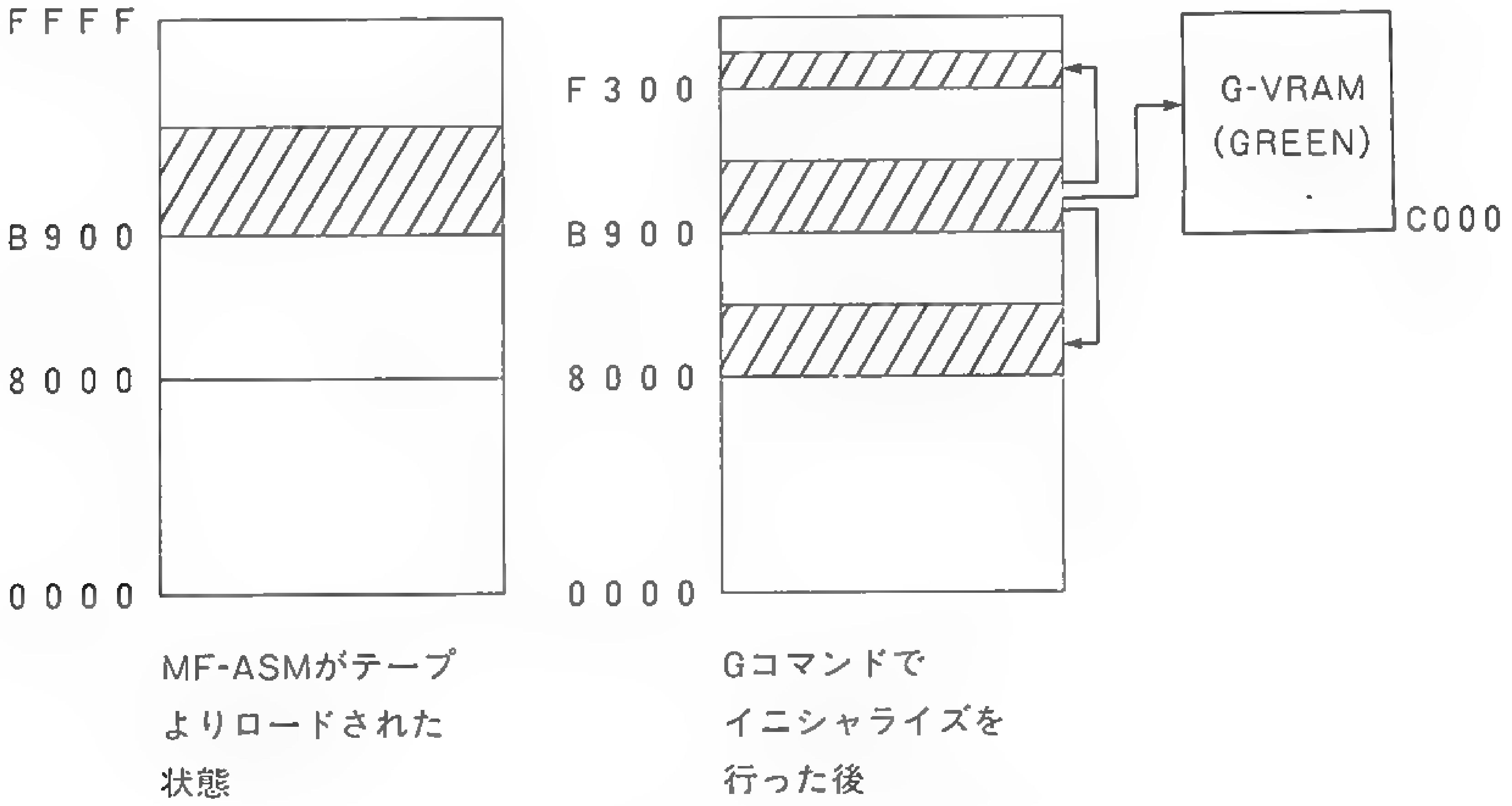
A．1．6　MF-ASMの格納状態

MF-ASMが最初にロードされた時点では，B900Hをスタート・アドレスとしてメモリにロードされています．

モニタ・モードに入り，GコマンドでB900H番地から実行すると以下の処理が行われます．

- テキスト・ウィンドウの実アドレス8000H～83FFH番地とワーク・エリア内にプログラムを転送する
- MF-ASM自身をG-VRAM(GREEN)へセーブする
- CMDの使用設定を行う

以下にMF-ASMがロードされたときとイニシャライズされたときのプログラムの格納状態を示します．


FFFF
B900
8000
0000
MF-ASMがテープよりロードされた状態
F300
B900
8000
0000
Gコマンドでイニシャライズを行った後
G-VRAM (GREEN)
C000

図A．2　MF-ASMの格納状態

A.2　MF-ASM文法

ここではMF-ASM文法について解説します．

A.2.1　文字

使用できる文字は，キャラクタコードの20H以上と，TABコードです．〝$〟や〝'〟は特殊な意味をもっています．

A.2.2　ステートメント(文)

ステートメントとは，文字の集まりで，ソース・プログラムの基本構成単位です．1ステートメントの最大文字数は80文字です．

A.2.3　ステートメントの構造

ステートメントの最初の文字が〝*〟または，〝;〟であれば，単なる注釈文(コメント文)として扱われます．〝*〟はアセンブル・リストにおいてTAB付けされませんが〝;〟はTAB付けされます．

注釈文以外の文は次の4つの部分から構成されますが，この4つの部分をすべてもつ必要はありません．

(1)　ラベル(名札宣言部)
(2)　命令(オペレーション)
(3)　オペランド
(4)　注釈(コメント)

ラベル : 命令　オペランド ; 注釈

- ラベル定義を行うときには，ラベルの直後に必ずコロン(:)を付ける
- 命令部とオペランドは1個以上のスペースで区切る
- 2つ以上のオペランドはカンマ(,)で区切る
- 命令部の前後やオペランド，数などの区切りにスペースはいくつあってもかまわない
- 実際には行番号とREM(')命令が行頭にある

A.2.4　ステートメント各部の説明

アセンブラでプログラムを記述する際の1つの単位としてまず「ネーム」という言葉の意味を定義しておきます．ネームとは，先頭の文字が〝?〟か〝@〟か英字(A～Z)で始まり，2文字以降が英数字(A～Z,0～9)で構成される6文字以内の文字列をいいます．

■ラベル

文の最初のネームの次の文字がコロンであれば，このネームはラベルとみなされ，その値はEQU命令のときはオペランドの値，それ以外はこのステートメントの先頭番地になります．EQU命令以外のときはコロンの後を省略することも可能です．ラベルとして次に示すレジスタ（レジスタ・ペア）名とコンディション・コード名は使用することができません．

A , B , C , D , E , H , L , AF , BC , DE , HL , IX , IY , SP , I , R , Z , C , NZ , NC , P , M , PE , PO

■オペレーション（命令）

ラベルでコロンがなかった場合やコロンの次にネームが存在した場合は，このネームは命令とみなされます．

命令は，μPD780の命令の他に，アセンブラに情報を与える擬似命令があります．命令は，ザイログニーモニックにほぼ準拠しています（ニーモニックについてはインストラクション表参照）．

■オペランド

命令が必要とする情報を与えます．命令によってはオペランドが不要のものがあります．

■注釈

セミコロン（；）のあとは注釈とみなされます．

A.2.5　擬似命令

擬似命令は，アセンブラに対して情報を与える命令で，以下のようなものがあります．

■ロケーション・カウンタ制御命令

ORG nn (Origin)

この命令以降，オペランドnnで指定されたアドレスからのマシン語コードを生成していきます．nnにラベルを含む場合には，あらかじめ定義されている必要があります．また，オペランドに式を記述することができます．

```
(例)  ORG   0E000H
      ORG    $+50
      ORG    START
```

■ シンボル定義命令

ラベル：EQU nn(Equate)

ラベルにnnの値を与えます．つまり，ラベルである定数を代表させることができEQUで定義した後は，そのラベルで値を記述することができます．nnにラベルを含んでいる場合にはあらかじめ定義されている必要があります．このステートメントには，必ずラベル部が存在しなければなりません．

```
(例) GVRAM0： EQU 5CH
     GVRAM2： EQU GVRAM0+1
     LF：     EQU 0AH
```

■ データ定義命令

データをプログラム中に設定したり，領域を確保したりするときに使用します．

▶DB n,… (Define Byte)

1バイト単位でデータを設定します．オペランドの値が，1バイトを超える大きさの場合は，その値の下位8ビットデータが，文字列のときは各文字がアスキー・コード（キャラクタ・コード）に変換されてプログラム中に格納されます．オペランドは，カンマで区切って複数設定できますが文字列は，30文字までです．オペランドにラベルがある場合には，それは定義されていなければなりません．

DEFB, DEFM, DCは，DBと同じです．

```
(例) DB 80H,30,LF+3
     DB 'MACHINE',0
     DB ''''
     DB 'CALL'+80H
     DB 1+2, 3+4
```

▶DW nn,‥‥ (Define Word)

オペランドのnnの値を16ビットのデータとして定義します．この場合，上位8ビットと下位8ビットが入れ換わることに注意してください．

文字列は最初の2文字だけが有効で，それ以後は無視されます．オペランドが1文字または1バイトで表現できる場合は，下位8ビットが0となります．オペランド中のラベルは定義されていなければなりません．

オペランドはカンマで区切って複数設定できます．

DEFWはDWと同じです

```
（例）DW   1234H
      DW   MSG,MSG+3
      DW   'AB'
      DW   1,2,3
```

▶DS nn, ‥‥ (Define Storage)

オペランドnnの値の大きさの領域をプログラム中にとります．すなわち，ロケーション・カウンタをnnだけ進めます．オペランドにラベルを含む場合，それは定義されていなければなりません．また確保した領域の内容は不定です．

DEFSはDSと同じです．

```
（例） DS  2
       DS BUFSIZ+1
       DS 2+2+2
       DS 2,2,2
```

■ アセンブル終了擬似命令

END(End)

アセンブラに対して，プログラムの終了を指示します．

END文以降に行があっても無視されます．

A.2.6　オペランド記述方法

オペランドには，命令が必要とするレジスタ名や式を書くことができます．式は項を演算子で結合したものでなければなりません．項には，定数と＄とラベルがあり，演算子には，+，−，と特殊演算子，R8があります．

▶定数

定数は，10進定数と16進定数および文字定数に分けることができます．16進数は，0～9とA～Fの16個の文字で表し，後にHをつけて10進定数と区別します．ただしA～Fで始まるときには，ラベルと区別するために前に0をつけなければなりません．

10進定数の場合，2バイトで表すことのできないあふれは無視されます．例えば，65537は1と同じことになります．

文字定数とは，引用符(')SHIFT-7で囲まれた1文字または2文字のことで，値はそのキャラクタ・コードになります．引用符(')自体を文字定数として使うときには，ダブルクウォ

ート(″)で1文字とします. コントロール・コード(キャリッジリターン・コード, 0DHなど)は, 10進定数または16進定数で記述します.

文字列は, DB命令のときに使用する部分は30文字以内で記述し, 30文字を超えるときは複数行に分けるようにします.

(例) 10進定数 0 123 −100

16進定数 0E000H 0DH .5CH 1234H

文字定数 'A''a''$'

▶$ (ロケーション・カウンタの値)

現在アセンブル中のステートメントの命令の番地を与えます.

(例)
```
JR      $
```

▶ラベル

そのラベルに定義された値を与えます.

(例)
```
ORG     START
DW      ADRES
LD      HL,DATA
```

▶＋

項と項との加算を行います.

▶−

項と項との減算を行います. また項の前に−が付くときはその項の2の補数がとられると考えます.

▶.R8

.R8より左に記述されている値16ビットを8ビットローテイトします. つまり, 上位8ビットと下位8ビットを入れ換えます.

▶特殊例

特殊例として,文字列定数との演算があります.これは, 文字定数または文字列定数の直後に演算子を続けるもので,引用符(')の中の最後の文字およびその前の文字との演算を行います.

(例)
```
'CALL'+80H
Z+1
AA+1111H
```

〈附則〉

(1) 相対ジャンプ(JR……,DJNZ)について

インストラクション表では，JRとDJNZは，そのオペランドに〝e〟というディスプレイスメント(変位置)をとることになっていますが，その際は次のようにアドレス(またはデータ)16ビットを記述することとし，ディスプレイスメントの計算はアセンブラが行います．

```
JR    nn
DJNZ  nn
```

```
(例)  JR    Z, 0E000H
      DJNZ  LOOP
      JR    NZ,$+2
```

(2) IN／OUT命令について

入出力命令は，次のように記述します．

```
IN      A,(n)
IN      r,(C)
OUT     (n),A
OUT     (C),r
```

(3) EX　AF, AF′について

EX AF, AF′はEX AF, AFのように第2オペランドもAFにします．

(4) オペランドの値について

オペランドに1バイトの値しか必要としない命令においては，オペランドの値の下位8ビットをデータとして扱います．

(例)　LD A,1234H→LD A,34Hと同じ

A.3　ユーティリティ・ソフトウェア

A.3.1　DUAD→MF-ASMソース・コンバータの使用法

このプログラムは，DUADのスクリーン・エディタを使用して作成し，ディスク上にセーブされたソース・プログラムをMF-ASM用のソース・プログラムに変換（つまり，REM文としてディスクにセーブ）するものです．

リスト6

```
100 '
110 '    convert DUAD88 source to MFASM source
120 '
130 CLEAR ,&HE5FF : DEFINT A-Z
140 DIM SOURCE.DATA$(100)
150 LINE INPUT "file name of DUAD88 source ? ";DUAD.NAME$
160 IF DUAD.NAME$="" THEN END
170 LINE INPUT "file name of MFASM source ? ";MFASM.NAME$
180 IF MFASM.NAME$="" THEN END
190 LINE INPUT "start line number ? ";L.NUMBER$
200 L.NUMBER=VAL(L.NUMBER$) : IF L.NUMBER=0 THEN L.NUMBER=100
210 '
220 OPEN DUAD.NAME$ FOR INPUT AS #1
230 OPEN MFASM.NAME$ FOR OUTPUT AS #2
240 '
250 WHILE NOT EOF(1)
260   I=0
270   WHILE I<=100 AND NOT EOF(1)
280     LINE INPUT #1,SOURCE.DATA$(I)
290     I=I+1
300   WEND
310 '
320   J=0
330   WHILE J<I
340     TEMP.DATA$=MID$(STR$(L.NUMBER),2)+" '"+SOURCE.DATA$(J)
350     PRINT TEMP.DATA$
360     PRINT #2,TEMP.DATA$
370     J=J+1 : L.NUMBER=L.NUMBER+10
380   WEND
390 WEND
400 CLOSE
410 END
```

RUNを実行すると，以下の３項目について質問してきます．

(1)　DUADのソース・プログラムのファイル・ネーム
(2)　ディスクにセーブするプログラムのファイル・ネーム
(3)　開始行番号

(1)はDUADのスクリーン・エディタで作成したソース・プログラムのファイル・ネームを答えます．拡張子はeとなります．

(2)は変換したプログラムをディスク上にセーブする際のファイル・ネームを指定します.

(3)はMF-ASMがBASICのREM文をソースとするため，BASICプログラムの開始行番号を指定します.

(1)，(2)を指定しなかった場合，プログラムはなにも行わずに終了します．(3)を指定しなかった場合は，開始行番号は100となります.

このプログラムの実行中は，変換したソース・リストがスクリーン上に表示されます.

変換終了後，変換したプログラムおよびMF-ASMをロードして実行すれば，マシン語プログラムが得られます.

A.3.2　MF-ASM→DUADソース・コンバータの使用法

このプログラムは，前出のプログラムの逆の変換を行うプログラムです．この変換により，MF-ASM用に作成したソース・プログラムをDUADのスクリーン・エディタの持つ豊富な機能を使用して編集することができるようになります.

リスト7

```
100 '
110 '   convert MFASM source to DUAD88 source
120 '
130 CLEAR ,&HE5FF : DEFINT A-Z
140 DIM SOURCE.DATA$(100)
150 LINE INPUT "file name of MFASM source ? ";MFASM.NAME$
160 IF MFASM.NAME$="" THEN END
170 LINE INPUT "file name of DUAD88 source ? ";DUAD.NAME$
180 IF DUAD.NAME$="" THEN END
190 '
200 OPEN MFASM.NAME$ FOR INPUT AS #1
210 OPEN DUAD.NAME$ FOR OUTPUT AS #2
220 '
230 WHILE NOT EOF(1)
240   I=0
250   WHILE I<=100 AND NOT EOF(1)
260     LINE INPUT #1,SOURCE.DATA$(I)
270     I=I+1
280   WEND
290 '
300   J=0
310   WHILE J<I
320     TEMP.DATA$=MID$(SOURCE.DATA$(J),INSTR(SOURCE.DATA$(J),"'")+1)
330     PRINT TEMP.DATA$
340     PRINT #2,TEMP.DATA$
350     J=J+1
360   WEND
370 WEND
380 CLOSE
390 END
```

RUNを実行すると，以下の2種類のファイル・ネームを質問してきますので，それぞれに答えてください．何も入力せずに RETURN を押すと，その場でプログラムの実行を終了します.

(1)　MF-ASMのソース・プログラムのファイル・ネーム
(2)　ディスクにセーブするプログラムのファイル・ネーム

このプログラムの実行に先立ち以下の点に注意してください．

- MF-ASM用のソース・プログラムを作成する際，REM文は必ず“'”(アポストロフィ)を使用する．“REM”であってはならない．
- ソース・プログラムをディスク上にセーブする場合，必ずアスキー形式でセーブする．

A．3．3　マシン語DATA文ジェネレータ

このプログラムは，メモリ上にロードされたマシン語プログラムをBASICのDATA文にするプログラムです．

本文中，USR関数とCALL文の説明文で例として出したBASICのプログラムでは，マシン語プログラムをデータとして持たせるためにこのプログラムを使用しました．でき上ったDATA文の使用方法については，それらのプログラムを参考にしてください．

RUNを実行すると以下のことを質問してきます．

(1)　マシン語プログラムのスタート・アドレス
(2)　マシン語プログラムのエンド・アドレス
(3)　出力するファイル・ネーム

(1)，(2)は必ず16進数で入力してください．指定しないとプログラムは何もせずに終了します．(3)の指定には，適当なファイル・ネームを指定すれば，行番号10000から始まるDATA文がディスク上にセーブされます．何も指定しなかった場合はDATA文がスクリーン上に表示されますから，カーソルを合わせて RETURN を押すことによってプログラムに DATA文を付加することができます．後からDATA文だけを残して他の部分をデリートすれば，目的とするマシン語のDATA文だけが得られます．

この方法を利用する場合，マシン語のプログラムサイズが大きくなりますと，1度に画面に表示できなくなりますので，分割してプログラムを実行してください．その際DATA文の開始行番号の指定は280行で行っていますので，適宜変更してください．

リスト8

```
100 '
110 '   data statement generater of machine program
120 '         default  screen
130 '
140 DEFINT A-Z
150 PRINT : PRINT
160 PRINT " ***  data statement generater of";
170 PRINT " machine language program ***"
```

```
180 PRINT
190 INPUT "    start address ";START.ADR$
200 INPUT "    end address   ";END.ADR$
210 IF START.ADR$="" OR END.ADR$="" THEN END
220 LINE INPUT "    output file name [SCRN:] ? ";FILE$
230 START.ADR=VAL("&h"+RIGHT$(START.ADR$,4))
240 END.ADR=VAL("&h"+RIGHT$(END.ADR$,4))
250 IF FILE$="" THEN FILE$="SCRN:"
260 '
270 OPEN FILE$ FOR OUTPUT AS #1
280   LINE.NUMBER=10000
290   FOR ADDRESS=START.ADR TO END.ADR STEP 16
300     WR$=MID$(STR$(LINE.NUMBER),2)+" DATA "
310     I=0
320     WHILE I < 15 AND ADDRESS+I < END.ADR
330       WR$=WR$+RIGHT$("0"+HEX$(PEEK(ADDRESS+I)),2)+","
340       I=I+1
350     WEND
360     WR$=WR$+RIGHT$("0"+HEX$(PEEK(ADDRESS+I)),2)
370     PRINT #1,WR$
380     LINE.NUMBER=LINE.NUMBER+10
390   NEXT ADDRESS
400 CLOSE
410 END
```

[注]　リスト6〜8はDISK-BASICでのみ利用可

リスト8はWHILE〜WENDをIF〜THENに変更すればROM-BASICでも利用できます.

A.4 Glibソース・プログラムの解析

ここに掲載されたGlibのソース・プログラムは，作者の好意により『PC-8801グラフィックスのすべて』より引用させていただいたものです．

このプログラムは，マシン語のプログラムとしては複雑なものですが，解析の一助となるように作者自身にコメントを付けていただきました．皆さんがこれを解析されれば，マシン語プログラミングの参考となると共に，PC-8801の新たな機能を発見できるでしょう．

```
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                 ;                                            ;
                 ;            PC-8801 Graphic Library1        ;
                 ;                                            ;
                 ;         SeP.  3 82' by cat soft (c)        ;
                 ;                                            ;
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                 ;
                 ;
8000             IMAGE: EQU  8000H                ; image and execute
                 ;     address
                 ;;; TEXT WINDOW OFFSET REGISTER DEPOSIT PORT ;;;
                 ;
                 ; read/write
                 ;
0070             PTWOFF:EQU  70H                  ; read/write
0078             PTWOFI:EQU  78H                  ; increment offset
                 ;
8390             IMAGEE:EQU  8390H
8000             IMAGEB:EQU  8000H
                 ;
                         ORG  0E000H
                 ;
E000 F3          BOOT:   DI                        割込みを禁止
E001 2118E0              LD   HL,BOOT9            ┐
E004 110080              LD   DE,IMAGE            ┝ブロック転送パラメータの設定
E007 019003              LD   BC,IMAGEE-IMAGEB    ┘
E00A DB70                IN   A,(PTWOFF)
E00C F5                  PUSH AF
E00D 3E80                LD   A,IMAGE.R8          ┐ブロック転送命令により，Glib の 8000H～83
E00F D370                OUT  (PTWOFF),A          ┘FFHに常駐する部分のプログラムを書き込む
E011 EDB0                LDIR
E013 F1                  POP  AF
E014 D370                OUT  (PTWOFF),A
E016 FB                  EI                        割込みを許可
E017 C9                  RET
                 ;
E018             BOOT9:  EQU  $
                 ;
E018                     END
```

```
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;                                        ;
                ;          PC-8801 Graphic Library2      ;
                ;                                        ;
                ;        Sep.  3 82' by cat soft (c)     ;
                ;                                        ;
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;
                ;
8000            IMAGE: EQU  8000H             ; image and execute
                ;    address
E000            LOAD:  EQU  0E000H            ; absolute load
                ;    location
                ;
                ; label definitions
                ;
C000            GRTOP: EQU  0C000H       G-VRAM先頭アドレス
FE80            GRBOT: EQU  0FE80H       G-VRAM表示最終アドレス+1
0050            GRWB:  EQU  80           G-VRAM横1行（640ドット）のバイト数
                ;
                ;;; keyboad scan port ;;;
                ;
0000            KBP0:  EQU  0           キーボードポート（テンキー7～0）
0001            KBP1:  EQU  1           キーボードポート（テンキー9～8）
0008            KBP8:  EQU  8           キーボードポート（ビット6がシフトキー）
                ;
                ;;; CRTC (mPD-3301) control prot ;;;
                ;
0050            CRTCP: EQU  50H         CRTC/パラメータポート
0051            CRTCC: EQU  51H         CRTCコマンドポート
                ;
                ;;; back ground & boarder color control port ;;;
                ;
                ; bit 6 : back ground color green
                ; bit 5 :         ,,        red
                ; bit 4 :         ,,        blue
                ;
                ; bit 2 : boarder color green          ,,
                ; bit 1 :         ,,    red            ,,
                ; bit 0 :         ,,    blue           ,,
                ;
0052            BCOLP: EQU  52H
                ;
                ;;; picture mixing port ;;;
                ;
                ; output : active low     そのビットが0のとき対応するVRAMのデータを表示
                ;
                ; bit 3 : G-VRAM2 display enable
                ; bit 2 : G-VRAM1         ,,
                ; bit 1 : G-VRAM0         ,,
                ; bit 0 : TEXT            ,,
                ;
                ;
0053            PMIXP: EQU  53H
                ;
                ;;; color pallet select port ;;;
                ;
                ; pallet 0 -- 7 is PORT 54H -- 5BH
                ;  bit 2 is green
                ;  bit 1 is red
                ;  bit 0 is blue
                ;
0054            PPAL:  EQU  54H
```

```
                ;
                ;;; graphic bank select port ;;;
                ;
                ; input
                ; G-VRAM status
                ; bit 2 : G-VRAM2 enable     現在表示されているグラフィック・バンクを表す
                ; bit 1 : G-VRAM1    ,,      そのビットが1のとき,対応するVRAMが表示されている
                ; bit 0 : G-VRAM0    ,,
                ;
005C            GRST:  EQU  5CH
                ;
                ; output
                ;
005C            GREB1: EQU  5CH              ; select bank#1
005D            GREB2: EQU  5DH              ;   ,,    bank#2
005E            GREB3: EQU  5EH              ;   ,,    bank#3
005F            GRDE:  EQU  5FH              ; select main bank
                ;
                ;;; kanji ROM port ;;;
                ;
                ; output
                ; KJALP  :                   漢字ROMに与えるアドレスの下位バイト
                ; KJAHP  :                   漢字ROMに与えるアドレスの上位バイト
                ; KJEP   :                   漢字ROMのデータ出力をイネイブルにする
                ; KJDP   :                   漢字ROMのデータ出力をディスエイブルにする
                ;
                ; !!! caution !!!
                ;  kanji ROM address is not equal to JIS code
                ;
00E8            KJALP: EQU  0E8H             データ出力がイネイブルのとき，このポートを読むと,
                                             漢字パターンデータの下位バイトが得られる
00E9            KJAHP: EQU  0E9H             データ出力がイネイブルのとき，このポートを読むと,
                                             漢字パターンデータの上位バイトが得られる
00EA            KJEP:  EQU  0EAH
00EB            KJDP:  EQU  0EBH
                ;
                       ORG  IMAGE
                ;
8000            IMAGEB:EQU  $
                ;
                ;;; put kanji pattern to graphic area ;;;
                ;                                                  〈コメント1ハイル〉
8000 2600       KPUTH: LD   H,0
8002 79                LD   A,C
8003 D60A              SUB  10                    ;200/20
8005 3802              JR   C,KPUTH1         640×400ドット・モード時に漢字表示位置より選択する
8007 4F                LD   C,A              グラフィック・バンクを算出する
8008 24                INC  H
8009 7C         KPUTH1:LD   A,H
                ;
800A C65C       KPUT:  ADD  A,GREB1
800C C5                PUSH BC
800D 4F                LD   C,A             Cレジスタに選択するグラフィック・バンクのI／Oアドレスを
                                            セット
800E EB                EX   DE,HL
800F 7C                LD   A,H
8010 FE30              CP   30H
8012 7D                LD   A,L
8013 3005              JR   NC,KPUT1
8015 1F                RRA
8016 E630              AND  30H
8018 1803              JR   KPUT2
801A 17         KPUT1: RLA                  非漢字（漢字コード2121H～2771H）か漢字（3021H～4F53
801B E6C0              AND  0C0H            H）かをチェックし，漢字コードを漢字ROMに与えるアドレ
801D 47         KPUT2: LD   B,A             スに交換する
801E 7D                LD   A,L
```

```
801F 17                     RLA
8020 17                     RLA
8021 17                     RLA
8022 E6F8                   AND   0F8H
8024 17                     RLA
8025 6F                     LD    L,A
8026 7C                     LD    A,H
8027 17                     RLA
8028 E63F                   AND   3FH
802A B0                     OR    B
802B 67                     LD    H,A
802C 227783                 LD    (KADR),HL      漢字ROMアドレスをKADRへ退避
                 ;
802F E1                     POP   HL
8030 CB24                   SLA   H
8032 5C                     LD    E,H
8033 7D                     LD    A,L
8034 17                     RLA
8035 85                     ADD   A,L
8036 17                     RLA
8037 C6C0                   ADD   A,GRTOP.R8     漢字表示位置より G-VRAM のアドレスを算出しHLレジス
8039 57                     LD    D,A            タにセット
803A AF                     XOR   A
803B CB3D                   SRL   L
803D 1F                     RRA
803E CB3D                   SRL   L
8040 1F                     RRA
8041 65                     LD    H,L
8042 6F                     LD    L,A
8043 19                     ADD   HL,DE
8044 0610                   LD    B,16
8046 ED79                   OUT   (C),A          グラフィック・バンクを選択する
                 ;
8048 EB          KPUT3:     EX    DE,HL
8049 2A7783                 LD    HL,(KADR)
804C 7C                     LD    A,H
804D D3E9                   OUT   (KJAHP),A      漢字ROMへアドレスを出力
804F 7D                     LD    A,L
8050 D3E8                   OUT   (KJALP),A
8052 23                     INC   HL
8053 227783                 LD    (KADR),HL
8056 AF                     XOR   A
8057 D3EA                   OUT   (KJEP),A       漢字ROMの出力をイネーブルにする
8059 DBE9                   IN    A,(KJAHP)
805B 67                     LD    H,A            漢字パターンをHLへロードする
805C DBE8                   IN    A,(KJALP)
805E 6F                     LD    L,A
805F AF                     XOR   A
8060 D3EB                   OUT   (KJDP),A       漢字ROMの出力をディスエーブルにする
8062 EB                     EX    DE,HL
8063 72                     LD    (HL),D         漢字パターンをG-VRAMへ描き込む
8064 23                     INC   HL
8065 73                     LD    (HL),E
8066 114F00                 LD    DE,GRWB-1
8069 19                     ADD   HL,DE
806A 10DC                   DJNZ  KPUT3
806C D35F                   OUT   (GRDE),A       メイン・バンクを選択する
806E C9                     RET
                 ;
                 ;;; clear screen all ;;;
                 ;                           ←〈コメント2ハイル〉
806F 3E02        CLS:       LD    A,2
8071 CD7A80                 CALL  CLSS
```

```
8074 3E01      CLSH:   LD    A,1
8076 CD7A80            CALL  CLSS
8079 AF        CLS1:   XOR   A
807A C65C      CLSS:   ADD   A,GREB1
807C 4F                LD    C,A
807D ED79              OUT   (C),A                グラフィック・バンクを選択する
807F 017F3E            LD    BC,GRBOT-GRTOP-1
8082 1101C0            LD    DE,GRTOP+1           ブロック転送命令（LDIR）により1画面分の
8085 2100C0            LD    HL,GRTOP             G-VRAMに0を描き込む
8088 3600              LD    (HL),0
808A EDB0              LDIR
808C D35F              OUT   (GRDE),A             メイン・バンクをセレクト
808E C9                RET
               ;
               ;;; roll up & down screen ;;;
               ;                                  ←〈コメント3ハイル〉
808F CD5B81    ROLUP:  CALL  GVRN
8092 2100C0            LD    HL,GRTOP
8095 19                ADD   HL,DE                ブロック転送のためのパラメータをセットする
8096 227983            LD    (ROLP),HL
8099 3E5C              LD    A,GREB1
809B CDCD80    ROLUP1:CALL   ROLUP2
809E 3C                INC   A                    グラフィック・バンクを0から2まで切り換えて画面を
809F FE5F              CP    GREB3+1              Roll up
80A1 20F8              JR    NZ,ROLUP1
80A3 C9                RET
               ;
80A4 CD5B81    ROLUPH:CALL   GVRN
80A7 2100C0            LD    HL,GRTOP
80AA 19                ADD   HL,DE                ブロック転送のためのパラメータをセットする
80AB 227983            LD    (ROLP),HL
80AE 3E5C              LD    A,GREB1
80B0 CDE280            CALL  ROLUP3               バンク0のRoll up
80B3 2100C0            LD    HL,GRTOP
80B6 ED4B8183          LD    BC,(ROLN)
80BA D35D      RLUPH1:OUT    (GREB2),A
80BC 7E                LD    A,(HL)
80BD EB                EX    DE,HL
80BE D35C              OUT   (GREB1),A
80C0 77                LD    (HL),A               バンク0とバンク1との間のデータ転送
80C1 EB                EX    DE,HL
80C2 23                INC   HL
80C3 13                INC   DE
80C4 0B                DEC   BC
80C5 78                LD    A,B
80C6 B1                OR    C
80C7 20F1              JR    NZ,RLUPH1
80C9 D35F              OUT   (GRDE),A
80CB 3E5D              LD    A,GREB2
               ;                                  バンク1のRoll up
80CD CDE280    ROLUP2:CALL   ROLUP3
80D0 62                LD    H,D
80D1 6B                LD    L,E
80D2 13                INC   DE
80D3 4F                LD    C,A
80D4 ED79              OUT   (C),A
80D6 3600              LD    (HL),0               ブロック転送を行ったバンク1の残りの領域を消去する
80D8 ED4B8183          LD    BC,(ROLN)
80DC 0B                DEC   BC
80DD EDB0              LDIR
80DF D35F              OUT   (GRDE),A
80E1 C9                RET
```

```
              ;
80E2 2A7983   ROLUP3:LD   HL,(ROLP)
80E5 1100C0          LD   DE,GRTOP
80E8 4F              LD   C,A         ACCの内容が示すグラフィック・バンクをブロック転送命令に
80E9 ED79            OUT  (C),A       よりRoll up
80EB ED4B7D83        LD   BC,(ROLC)
80EF EDB0            LDIR
80F1 D35F            OUT  (GRDE),A
80F3 C9              RET
              ;
              ;
80F4 CD5B81   ROLDW: CALL GVRN
80F7 1100C0          LD   DE,GRTOP
80FA 19              ADD  HL,DE       ブロック転送のためのパラメータをセットする
80FB 2B              DEC  HL
80FC 227983          LD   (ROLP),HL
80FF 3E5E            LD   A,GREB3
8101 CD3481   ROLDW1:CALL ROLDW2
8104 3D              DEC  A           バンクを2から0まで切り換えて各バンクをRoll down
8105 FE5B            CP   GREB1-1
8107 20F8            JR   NZ,ROLDW1
8109 C9              RET
              ;
              ;
810A CD5B81   ROLDWH:CALL GVRN
810D 1100C0          LD   DE,GRTOP
8110 19              ADD  HL,DE       ブロック転送のためのパラメータをセットする
8111 2B              DEC  HL
8112 227983          LD   (ROLP),HL
8115 3E5D            LD   A,GREB2     バンク1をブロック転送命令によりRoll down
8117 CD4981          CALL ROLDW3
811A 217FFE          LD   HL,GRBOT-1
811D ED4B8183        LD   BC,(ROLN)
8121 D35C     RLDWH1:OUT  (GREB1),A
8123 7E              LD   A,(HL)
8124 EB              EX   DE,HL
8125 D35D            OUT  (GREB2),A
8127 77              LD   (HL),A      バンク0からバンク1へのブロック転送
8128 EB              EX   DE,HL
8129 2B              DEC  HL
812A 1B              DEC  DE
812B 0B              DEC  BC
812C 78              LD   A,B
812D B1              OR   C
812E 20F1            JR   NZ,RLDWH1
8130 D35F            OUT  (GRDE),A
8132 3E5C            LD   A,GREB1     バンク0のRoll down
              ;
8134 CD4981   ROLDW2:CALL ROLDW3
8137 62              LD   H,D
8138 6B              LD   L,E
8139 1B              DEC  DE
813A 4F              LD   C,A
813B ED79            OUT  (C),A       ブロック転送を行ったバンク0の残りの領域を消去する
813D 3600            LD   (HL),0
813F ED4B8183        LD   BC,(ROLN)
8143 0B              DEC  BC
8144 EDB8            LDDR
8146 D35F            OUT  (GRDE),A
8148 C9              RET
              ;
```

```
8149 2A7983    ROLDW3:LD    HL,(ROLP)
814C 117FFE          LD    DE,GRBOT-1
814F 4F              LD    C,A
8150 ED79            OUT   (C),A        ACCの示すグラフィック・バンクをブロック転送命令により
8152 ED4B7D83        LD    BC,(ROLC)    Roll down
8156 EDB8            LDDR
8158 D35F            OUT   (GRDE),A
815A C9              RET
               ;
               ;
815B A7        GVRN:  AND   A
815C 281D            JR    Z,GVRN1      Roll up, Roll downのパラメータのチェック
815E FEC8            CP    200
8160 3019            JR    NC,GVRN1
8162 2600            LD    H,0
8164 6F              LD    L,A
8165 54              LD    D,H
8166 5F              LD    E,A
8167 29              ADD   HL,HL
8168 29              ADD   HL,HL
8169 19              ADD   HL,DE
816A 29              ADD   HL,HL          ブロック転送のパラメータの算出
816B 29              ADD   HL,HL
816C 29              ADD   HL,HL
816D 29              ADD   HL,HL
816E 228183          LD    (ROLN),HL
8171 EB              EX    DE,HL
8172 21803E          LD    HL,GRBOT-GRTOP
8175 ED52            SBC   HL,DE
8177 227D83          LD    (ROLC),HL
817A C9              RET
817B 33        GVRN1: INC   SP
817C 33              INC   SP
817D C9              RET
               ;
               ;;; roll left & right screen ;;;
               ;
817E CD1382    ROLLI: CALL  GHRN          ←〈コメント4ハイル〉
8181 3E5E            LD    A,GREB3
8183 1805            JR    ROLLI1
               ;
8185 CD1382    ROLLIH:CALL  GHRN
8188 3E5D            LD    A,GREB2
818A 2100C0    ROLLI1:LD    HL,GRTOP
818D 19              ADD   HL,DE
818E 227983          LD    (ROLP),HL
8191 2150C0          LD    HL,GRTOP+GRWB
8194 ED52            SBC   HL,DE
8196 227B83          LD    (ROLQ),HL
8199 2A7983    ROLLI2:LD    HL,(ROLP)
819C 1100C0          LD    DE,GRTOP      ACCの内容が示すグラフィック・バンクをブロック転送によ
819F 4F              LD    C,A           りRoll leftを行う
81A0 ED79            OUT   (C),A
81A2 ED4B7D83        LD    BC,(ROLC)
81A6 EDB0            LDIR
81A8 328383          LD    (ROLWK),A
81AB 2A7F83          LD    HL,(ROLM)
81AE EB              EX    DE,HL
81AF 2A7B83          LD    HL,(ROLQ)
81B2 0EC8            LD    C,200
81B4 3A8183          LD    A,(ROLN)
81B7 47        ROLLI3:LD    B,A           ブロック転送により右側にまわり込んで表示されている図形を
81B8 3600      ROLLI4:LD    (HL),0        消去する
```

```
81BA 23                 INC   HL
81BB 10FB               DJNZ  ROLLI4
81BD 19                 ADD   HL,DE
81BE 0D                 DEC   C
81BF 20F6               JR    NZ,ROLLI3
81C1 3A8383             LD    A,(ROLWK)
81C4 3D                 DEC   A
81C5 FE5B               CP    GREB1-1
81C7 20D0               JR    NZ,ROLLI2
81C9 D35F               OUT   (GRDE),A
81CB C9                 RET
               ;
81CC CD1382    ROLRI:   CALL  GHRN
81CF 3E5E               LD    A,GREB3
81D1 1805               JR    ROLRI1
               ;
81D3 CD1382    ROLRIH:CALL  GHRN
81D6 3E5D               LD    A,GREB2
81D8 217FFE    ROLRI1:LD    HL,GRBOT-1
81DB ED52               SBC   HL,DE
81DD 227983             LD    (ROLP),HL
81E0 2A7983    ROLRI2:LD    HL,(ROLP)     ACCの内容が示すグラフィック・バンクをブロック転送命令
81E3 117FFE             LD    DE,GRBOT-1    によりRoll right する
81E6 4F                 LD    C,A
81E7 ED79               OUT   (C),A
81E9 ED4B7D83           LD    BC,(ROLC)
81ED EDB8               LDDR
81EF 328383             LD    (ROLWK),A
81F2 2A7F83             LD    HL,(ROLM)
81F5 EB                 EX    DE,HL
81F6 2100C0             LD    HL,GRTOP
81F9 0EC8               LD    C,200
81FB 3A8183             LD    A,(ROLN)
81FE 47        ROLRI3:LD    B,A           ブロック転送により左側にまわり込んで表示されている図形を
81FF 3600      ROLRI4:LD    (HL),0        消去する
8201 23                 INC   HL
8202 10FB               DJNZ  ROLRI4
8204 19                 ADD   HL,DE
8205 0D                 DEC   C
8206 20F6               JR    NZ,ROLRI3
8208 3A8383             LD    A,(ROLWK)
820B 3D                 DEC   A
820C FE5B               CP    GREB1-1
820E 20D0               JR    NZ,ROLRI2
8210 D35F               OUT   (GRDE),A
8212 C9                 RET
               ;
8213 210700    GHRN:    LD    HL,7
8216 19                 ADD   HL,DE
8217 7D                 LD    A,L
8218 CB3C               SRL   H
821A 1F                 RRA
821B CB3C               SRL   H
821D 1F                 RRA
821E CB3C               SRL   H
8220 1F                 RRA
8221 2600               LD    H,0
8223 6F                 LD    L,A             左右のRollのスクロール幅よりブロック転送のための
8224 228183             LD    (ROLN),HL       パラメータを算出する
8227 EB                 EX    DE,HL
8228 215000             LD    HL,GRWB
822B A7                 AND   A
822C ED52               SBC   HL,DE
```

```
822E 227F83            LD    (ROLM),HL
8231 21803E            LD    HL,GRBOT-GRTOP
8234 ED52              SBC   HL,DE
8236 227D83            LD    (ROLC),HL
8239 C9                RET
                 ;
                 ;;; move and display graphic cursor ;;;
                                                       ←〈コメント5ハイル〉
                 ;
0001             STEP1: EQU   1                   ; unshifted step
0008             STEP2: EQU   8                   ; shifted step
                 ;
823A ED538583    GCREV: LD    (GX),DE       クロス・カーソルの表示範囲を設定する
823E ED438783           LD    (GY),BC
8242 DD228983           LD    (GXM),IX      クロス・カーソルの表示位置を設定する
8246 FD228B83           LD    (GYM),IY
824A CD3483             CALL  VLINE         クロス・カーソルを表示する
824D C3F882             JP    HLINE
                 ;
8250 AF          GCMOVE:XOR   A             GCVALをクリアする
8251 328F83             LD    (GCVAL),A
8254 218D83             LD    HL,GS
8257 3608               LD    (HL),STEP2    シフトキーのスキャンを行い1回のクロス・カーソルの移動
8259 DB08               IN    A,(KBP8)      距離をGSにセット(シフト時8ドット,アンシフト時1ドット)
825B E640               AND   40H                 ; scan shift key
825D 2802               JR    Z,KBSCAN
825F 3601               LD    (HL),STEP1
8261 DB00        KBSCAN:IN    A,(KBP0)
8263 2F                 CPL
8264 57                 LD    D,A           テンキー0～9をスキャンし読み取ったデータを反転してDE
8265 DB01               IN    A,(KBP1)      レジスタへロードする
8267 2F                 CPL
8268 5F                 LD    E.A
                 ;
8269 010000             LD    BC,0                ; dx=dy=0;△x，△yを0とする
826C E602               AND   02H                 ; scan 9
826E 2005               JR    NZ,KBSC1                        3，6，9のキーが押され
8270 7A                 LD    A,D                             ている場合には△x＝1と
8271 E648               AND   48H                 ; scan 36   する
8273 2801               JR    Z,KBSC2
8275 04          KBSC1: INC   B
8276 7A          KBSC2: LD    A,D                             1，4，7のキーが押され
8277 E692               AND   92H                 ; scan 147  ている場合には△x＝△x
8279 2801               JR    Z,KBSC3                         －1とする
827B 05                 DEC   B
827C 7A          KBSC3: LD    A,D                             1，2，3のキーが押され
827D E60E               AND   0EH                 ; scan 123  ている場合には△y＝1
827F 2801               JR    Z,KBSC4                         とする
8281 0C                 INC   C
8282 7A          KBSC4: LD    A,D
8283 E680               AND   80H                 ; scan 7
8285 2005               JR    NZ,KBSC5                        7、8，9のキーが押され
8287 7B                 LD    A,E                             ている場合には△y＝△y
8288 E603               AND   03H                 ; scan 89   －1とする
828A 2801               JR    Z,KBSC6
828C 0D          KBSC5: DEC   C
828D 78          KBSC6: LD    A,B
828E B1                 OR    C
828F C49E82             CALL  NZ,GCMOV2
8292 ED5B8583           LD    DE,(GX)      カーソルの水平位置をHLレジスタ・ペアへセット
8296 ED4B8783           LD    BC,(GY)      カーソルの垂直位置をBCレジスタ・ペアへセット
829A 3A8F83             LD    A,(GCVAL)    カーソルが移動したか否かのフラグをACCへロードする
829D C9                 RET
```

```
                 ;
                 ;
829E 3A8F83      GCMOV2:LD    A,(GCVAL)
82A1 F6FFI              OR    0FFH         GCVAL（カーソル移動フラグ）にFFHをセットする
82A3 328F83             LD    (GCVAL),A
82A6 78                 LD    A,B
82A7 2A8D83             LD    HL,(GS)
82AA EB                 EX    DE,HL
82AB 210000             LD    HL,0
82AE A7                 AND   A
82AF 2815               JR    Z,GCMOV4
82B1 FAB582             JP    M,GCMOV3
82B4 EB                 EX    DE,HL
82B5 ED52        GCMOV3:SBC   HL,DE
82B7 EB                 EX    DE,HL
82B8 2A8583             LD    HL,(GX)
82BB 19                 ADD   HL,DE
82BC EB                 EX    DE,HL        新しいカーソルの水平位置を算出しHLレジスタへセットする
82BD 2A8983             LD    HL,(GXM)
82C0 A7                 AND   A
82C1 ED52               SBC   HL,DE
82C3 EB                 EX    DE,HL
82C4 3003               JR    NC,GCMOV5
82C6 2A8583      GCMOV4:LD    HL,(GX)
82C9 C5          GCMOV5:PUSH  BC
82CA E5                 PUSH  HL
82CB CD3483             CALL  VLINE        現在の垂直カーソルを消去する
82CE E1                 POP   HL
82CF 228583             LD    (GX),HL      新しい水平位置をGXへセットする
82D2 CD3483             CALL  VLINE        新しい垂直カーソルのG-VRAMへ描き込み
82D5 C1                 POP   BC
                 ;
82D6 3A8B83             LD    A,(GYM)
82D9 3C                 INC   A
82DA 47                 LD    B,A
82DB 79                 LD    A,C
82DC A7                 AND   A
82DD 3A8D83             LD    A,(GS)
82E0 4F                 LD    C,A
82E1 3A8783             LD    A,(GY)
82E4 280A               JR    Z,GCMOV8     新しいカーソルの垂直位置を算出しACCへセットする
82E6 FAED82             JP    M,GCMOV7
82E9 81          GCMOV6:ADD   A,C
82EA B8                 CP    B
82EB 3803               JR    C,GCMOV8
                 ;
82ED 91          GCMOV7:SUB   C
82EE 38F9               JR    C,GCMOV6
82F0 F5          GCMOV8:PUSH  AF
82F1 CDF882             CALL  HLINE        現在の水平カーソルを消去する
82F4 F1                 POP   AF
82F5 328783             LD    (GY),A       新しい垂直位置をGYにセットする
                 ;
82F8 2A8783      HLINE: LD    HL,(GY)
82FB 29                 ADD   HL,HL
82FC 29                 ADD   HL,HL
82FD 29                 ADD   HL,HL
82FE 29                 ADD   HL,HL
82FF 54                 LD    D,H
8300 5D                 LD    E,L
8301 29                 ADD   HL,HL
8302 29                 ADD   HL,HL
8303 19                 ADD   HL,DE
```

```
8304 1100C0            LD   DE,GRTOP
8307 19                ADD  HL,DE
8308 E5                PUSH HL
8309 ED4B8983          LD   BC,(GXM)
830D 79                LD   A,C        垂直位置より水平カーソルを描き込むグラフィック・バンクのアド
830E CB38              SRL  B          レスを算出する
8310 CB19              RR   C
8312 CB38              SRL  B
8314 CB19              RR   C
8316 CB38              SRL  B
8318 CB19              RR   C
831A E607              AND  7
831C 1600              LD   D,0
831E 5F                LD   E,A
831F 216783            LD   HL,HBITT
8322 19                ADD  HL,DE
8323 41                LD   B,C
8324 4E                LD   C,(HL)
8325 E1                POP  HL
8326 D35C              OUT  (GREB1),A バンク0を選択する
8328 7E        HLINE1:LD   A,(HL)
8329 2F                CPL
832A 77                LD   (HL),A
832B 23                INC  HL
832C 10FA              DJNZ HLINE1     水平カーソルをG-VRAMへ描き込む
832E 79                LD   A,C
832F AE                XOR  (HL)
8330 77                LD   (HL),A
8331 D35F              OUT  (GRDE),A   メイン・バンクを選択する
8333 C9                RET
               ;
8334 2A8583    VLINE: LD   HL,(GX)
8337 7D                LD   A,L
8338 CB3C              SRL  H
833A CB1D              RR   L
833C CB3C              SRL  H
833E CB1D              RR   L
8340 CB3C              SRL  H
8342 CB1D              RR   L
8344 1100C0            LD   DE,GRTOP
8347 19                ADD  HL,DE
8348 EB                EX   DE,HL      水平位置より垂直カーソルを描き込むグラフィック・バンクのア
8349 216F83            LD   HL,VBITT   ドレスを算出する
834C E607              AND  7
834E 0600              LD   B,0
8350 4F                LD   C,A
8351 09                ADD  HL,BC
8352 4E                LD   C,(HL)
8353 EB                EX   DE,HL
8354 115000            LD   DE,GRWB
8357 3A8B83            LD   A,(GYM)
835A 3C                INC  A
835B 47                LD   B,A
835C D35C              OUT  (GREB1),Aバンクの0を選択する
835E 79        VLINE1:LD   A,C
835F AE                XOR  (HL)
8360 77                LD   (HL),A     垂直カーソルをG-VRAMへ描き込む
8361 19                ADD  HL,DE
8362 10FA              DJNZ VLINE1
8364 D35F              OUT  (GRDE),A   メイン・バンクを選択する
8366 C9                RET
```

```
                ;
                ;
8367 80C0E0F0 HBITT: DEFB 80H,0C0H,0E0H,0F0H        }水平カーソルのためのデータ・テー
836B F8FCFEFF        DEFB 0F8H,0FCH,0FEH,0FFH       }ブル
836F 80402010 VBITT: DEFB 80H,40H,20H,10H,8,4,2,1 垂直カーソルのためのデータ・テーブル
8373 08040201
                ;
8377 0000       KADR:  DEFW 0               漢字ROMアドレスの退避領域
                ;
8379 0000       ROLP:  DEFW 0               ┐
837B 0000       ROLQ:  DEFW 0               │
837D 0000       ROLC:  DEFW 0               │
837F 0000       ROLM:  DEFW 0               ├Roll で用いるブロック転送のパラメータ領域
8381 0000       ROLN:  DEFW 0               │
8383 0000       ROLWK: DEFW 0               ┘
                ;
8385 4001       GX:    DEFW 320             ┐
8387 6300       GY:    DEFW 99              │
8389 7F02       GXM:   DEFW 639             │
838B C700       GYM:   DEFW 199             ├クロス・カーソルの表示位置及び表示範囲と1回の移動距離
838D 0100       GS:    DEFW STEP1           │
838F 00         GCVAL: DEFB 0               ┘
                ;
8390            IMAGEE:EQU  $
                ;
8390                   END
                ;
E49F            PARM4:                               ; for GCREV
E49F 2AFCE5            LD   HL,(ARG2)
E4A2 CDC0E4            CALL PARMG    ┐
E4A5 DD210000          LD   IX,0     │
E4A9 DD19              ADD  IX,DE    │
                ;                    │
E4AB 2AFEE5            LD   HL,(ARG3)├ARG2をIXへ, ARG3をIYへセットする
E4AE CDC0E4            CALL PARMG    │
E4B1 FD210000          LD   IY,0     │
E4B5 FD19              ADD  IY,DE    ┘
                ;
E4B7            PARM3:                               ; for PROT,PROTX,PROTY
E4B7 2AFAE5            LD   HL,(ARG1)┐
E4BA 4E                LD   C,(HL)   │
E4BB 23                INC  HL       │
E4BC 46                LD   B,(HL)   │
                ;                    ├ARG1をBCへ, ARG0をDEへセットする
E4BD            PARM2:               │       ; for ROLLI,ROLLIH,ROLRI,
E4BD 2AF8E5            LD   HL,(ARG0)│         ROLRIH
E4C0 5E         PARMG: LD   E,(HL)   │
E4C1 23                INC  HL       ┘
E4C2 56                LD   D,(HL)
E4C3 C9                RET
                ;
E4C4 2AFCE5     KPUTL: LD   HL,(ARG2)┐
E4C7 7E                LD   A,(HL)   │
E4C8 210A80            LD   HL,KPUT  ├漢字を描き込むグラフィック・バンクをACCにセットする
E4CB 1803              JR   KPUTL1   ┘
E4CD 210080     KPUTLH:LD   HL,KPUTH
E4D0 32DAE5     KPUTL1:LD   (KBANK),A
E4D3 22DDE5            LD   (KVECT),HL
E4D6 2AFAE5            LD   HL,(ARG1)
E4D9 6E                LD   L,(HL)
E4DA 26FF              LD   H,0FFH
E4DC 22DBE5            LD   (KLOC),HL
E4DF 2AF8E5            LD   HL,(ARG0)
```

```
E4E2 5E                  LD   E,(HL)
E4E3 23                  INC  HL
E4E4 56                  LD   D,(HL)
E4E5 EB                  EX   DE,HL
E4E6 7E                  LD   A,(HL)
E4E7 23                  INC  HL
E4E8 D602                SUB  2
E4EA 2824                JR   Z,KPUTL3
E4EC 5E                  LD   E,(HL)
E4ED 23                  INC  HL
E4EE 56                  LD   D,(HL)
E4EF EB                  EX   DE,HL
E4F0 23                  INC  HL
E4F1 23                  INC  HL
E4F2 1F                  RRA
E4F3 47                  LD   B,A
E4F4 C5         KPUTL2:PUSH BC
E4F5 ED4BDBE5            LD   BC,(KLOC)
E4F9 04                  INC  B
E4FA ED43DBE5            LD   (KLOC),BC
E4FE 5E                  LD   E,(HL)
E4FF 23                  INC  HL
E500 56                  LD   D,(HL)
E501 23                  INC  HL          ストリングで与えた漢字文字列をG-VRAMへ描き込む
E502 3ADAE5              LD   A,(KBANK)
E505 E5                  PUSH HL
E506 2ADDE5              LD   HL,(KVECT)
E509 CD51E4              CALL JUMP1
E50C E1                  POP  HL
E50D C1                  POP  BC
E50E 10E4                DJNZ KPUTL2
E510 ED4BDBE5   KPUTL3:LD   BC,(KLOC)
E514 2ADDE5              LD   HL,(KVECT)
E517 04         KPUTL4:INC  B
E518 78                  LD   A,B
E519 FE28                CP   40
E51B 300F                JR   NC,KPUTL5
E51D 112121              LD   DE,KSPACE   一行のうち漢字を描き込んだ残りの領域に空白を描き込む
E520 3ADAE5              LD   A,(KBANK)
E523 C5                  PUSH BC
E524 E5                  PUSH HL
E525 CD51E4              CALL JUMP1
E528 E1                  POP  HL
E529 C1                  POP  BC
E52A 18EB                JR   KPUTL4
E52C C9         KPUTL5:RET
                ;
E52D            PARM5:                          ; for GCMOVE return value
E52D 2AF8E5              LD   HL,(ARG0)
E530 73                  LD   (HL),E
E531 23                  INC  HL          移動後のクロスカーソルの水平位置をARG 0の示す変数へ,
E532 72                  LD   (HL),D      垂直位置をARG1の示す変数へそれぞれセットする
E533 2AFAE5              LD   HL,(ARG1)
E536 71                  LD   (HL),C
E537 23                  INC  HL
E538 70                  LD   (HL),B
E539 32E3E5              LD   (USRVAL),A  クロス・カーソルが移動したか否かのフラグをUSRVALへ
E53C C9                  RET              セットする
                ;
E53D            SVOFF:                          ; test window offset value
E53D 00                  DB   0                   save ar
```

```
                   ;
                   ;;; convart pattern x <--> y ;;;
                   ;                                ← 〈コメント6ハイル〉
E53E CDB6E5        PROT:   CALL GETLEN              配列よりパターンの大きさ等のデータを取り出す
E541 FD2ADFE5              LD   IY,(PRP)
E545 47                    LD   B,A
E546 5F                    LD   E,A
E547 C5            PROT1:  PUSH BC
E548 2E80                  LD   L,80H
E54A FD22DFE5      PROT2:  LD   (PRP),IY
E54E 4B                    LD   C,E
E54F 0608          PROT3:  LD   B,8
E551 DD6600                LD   H,(IX)
E554 DD23                  INC  IX
E556 7D            PROT4:  LD   A,L
E557 CB24                  SLA  H
E559 3005                  JR   NC,PROT5
E55B FDB600                OR   (IY)                オリジナル・パターンより1バイト取り出し，上下方向にビット列を
E55E 1804                  JR   PROT6               展開して配列内にプロットする
E560 2F            PROT5:  CPL
E561 FDA600                AND  (IY)
E564 FD7700        PROT6:  LD   (IY),A
E567 FD19                  ADD  IY,DE
E569 10EB                  DJNZ PROT4
E56B 0D                    DEC  C
E56C 20E1                  JR   NZ,PROT3
E56E FD2ADFE5              LD   IY,(PRP)
E572 CB3D                  SRL  L
E574 30D4                  JR   NC,PROT2
E576 FD23                  INC  IY
E578 C1                    POP  BC
E579 10CC                  DJNZ PROT1
E57B C9                    RET
                   ;
E57C CDB6E5        PROTX:  CALL GETLEN              配列よりパターンの大きさ等のデータを取り出す
E57F 4B                    LD   C,E
E580 5F                    LD   E,A
E581 19            PROTX1:ADD   HL,DE
E582 43                    LD   B,E
E583 DD7E00        PROTX2:LD    A,(IX)
E586 DD23                  INC  IX
E588 C5                    PUSH BC
E589 0608                  LD   B,8
E58B 4F                    LD   C,A
E58C CB21          PROTX3:SLA   C                   横1列分のパターンをx軸方向でビット列を反転し，配列内にプロ
E58E 1F                    RRA                      ットする
E58F 10FB                  DJNZ PROTX3
E591 C1                    POP  BC
E592 2B                    DEC  HL
E593 77                    LD   (HL),A
E594 10ED                  DJNZ PROTX2
E596 0D                    DEC  C
E597 C8                    RET  Z
E598 19                    ADD  HL,DE
E599 18E6                  JR   PROTX1
                   ;
E59B CDB6E5        PROTY:  CALL GETLEN              配列よりパターンの大きさ等のデータを取り出す
E59E 47                    LD   B,A
E59F 19            PROTY1:ADD   HL,DE
E5A0 10FD                  DJNZ PROTY1
E5A2 4B                    LD   C,E
E5A3 5F                    LD   E,A
E5A4 ED52          PROTY2:SBC   HL,DE
```

```
E5A6 43                LD   B,E
E5A7 DD7E00     PROTY3:LD   A,(IX)      ┐
E5AA DD23              INC  IX          │横１列分のパターンのデータをｙ軸方向の位置を変えて，１列分
E5AC 77                LD   (HL),A      │転送する
E5AD 23                INC  HL          │
E5AE 10F7              DJNZ PROTY3      ┘
E5B0 0D                DEC  C
E5B1 C8                RET  Z
E5B2 ED52              SBC  HL,DE
E5B4 18EE              JR   PROTY2
                ;
E5B6 D5         GETLEN:PUSH DE
E5B7 DDE1              POP  IX
E5B9 AF                XOR  A
E5BA 60                LD   H,B
E5BB 69                LD   L,C
E5BC DD4E00            LD   C,(IX)
E5BF 71                LD   (HL),C
E5C0 23                INC  HL
E5C1 77                LD   (HL),A
E5C2 23                INC  HL
E5C3 DD4602            LD   B,(IX+2)
E5C6 70                LD   (HL),B
E5C7 23                INC  HL
E5C8 77                LD   (HL),A
E5C9 23                INC  HL
E5CA 22DFE5            LD   (PRP),HL    反転したパターンを書き込む領域の先頭アドレスをＰＲＰにセッ
                ;                       トする
E5CD 110400            LD   DE,4        ｝オリジナルパターンのデータに領域の先頭アドレスをＩＸにセッ
E5D0 DD19              ADD  IX,DE       ｝トする
                ;
E5D2 58                LD   E,B         パターンの一辺の大きさ(ドット数)をＥ，Ａのレジスタへセットす
E5D3 78                LD   A,B         る
E5D4 0F                RRCA
E5D5 0F                RRCA
E5D6 0F                RRCA             ┐
E5D7 E61F              AND  1FH         ｝パターンの一辺の大きさ（バイト数）をＡＣＣにセットする
E5D9 C9                RET              ┘


                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;                                         ;
                ;        PC-8801 Graphic Library3         ;
                ;                                         ;
                ;      Sep.  3 82´ by cat soft (c)        ;
                ;                                         ;
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;
                ;
8000            IMAGE: EQU  8000H
                ;
                ;;; text window offset register deposit port ;;;
                ;
                ; read/write
                ;                                            テキスト・
0070            PTWOFF:EQU  70H          ; read/write      ┐ウィンド
0078            PTWOFI:EQU  78H          ; increment offset｝ウのオフ
                ;                                          ┘セット・
                                                            レジスタ
2121            KSPACE:EQU  2121H          空白文字の漢字コード
                ;
                ;;; DEFINE AT IMAGE PROGRAM;;;
806F            CLS:   EQU  806FH
807A            CLSS:  EQU  807AH
```

```
8074            CLSH:   EQU   8074H
808F            ROLUP:  EQU   808FH
80A4            ROLUPH:EQU    80A4H
80F4            ROLDW:  EQU   80F4H
810A            ROLDWH:EQU    810AH
817E            ROLLI:  EQU   817EH
8185            ROLLIH:EQU    8185H
81CC            ROLRI:  EQU   81CCH
81D3            ROLRIH:EQU    81D3H
823A            GCREV:  EQU   823AH
8250            GCMOVE:EQU    8250H
                ;
800A            KPUT:   EQU   800AH
8000            KPUTH:  EQU   8000H
                ;
                        ORG   0E400H
                ;
E400 F3         ENTRY:  DI                     Glibを呼び出す場合のエントリーポイント
E401 FE02               CP    2                USR関数の引数が整数型か否かチェックする
E403 202F               JR    NZ,EXIT          整数型でない場合には何もしない
E405 22E1E5             LD    (VARPTR),HL
E408 AF                 XOR   A
E409 32E3E5             LD    (USRVAL),A
E40C 23                 INC   HL
E40D 7E                 LD    A,(HL)
E40E A7                 AND   A
E40F 2023               JR    NZ,EXIT
E411 57                 LD    D,A             }USR関数の引数の値を取り出し，チェックする
E412 2B                 DEC   HL
E413 7E                 LD    A,(HL)
E414 FE13               CP    LIBMAX+1
E416 301C               JR    NC,EXIT
E418 87                 ADD   A,A
E419 87                 ADD   A,A
E41A 5F                 LD    E,A
E41B 2152E4             LD    HL,JMPTBL
E41E 19                 ADD   HL,DE
E41F 5E                 LD    E,(HL)          }引数を4倍してジャンプ·テーブルよりアドレスを取り出し，
E420 23                 INC   HL               DE，BCレジスタにセットする
E421 56                 LD    D,(HL)
E422 23                 INC   HL
E423 4E                 LD    C,(HL)
E424 23                 INC   HL
E425 46                 LD    B,(HL)
E426 C5                 PUSH  BC
E427 CD46E4             CALL  SELBNK           テキスト・ウィンドウの切り換え
E42A CD50E4             CALL  JUMP
E42D E1                 POP   HL
E42E CD51E4             CALL  JUMP1
E431 CD40E4             CALL  DSLBNK           テキスト・ウィンドウをもとに戻す
E434 2AE1E5     EXIT:   LD    HL,(VARPTR)
E437 3AE3E5             LD    A,(USRVAL)
E43A 77                 LD    (HL),A
E43B 23                 INC   HL
E43C 3600               LD    (HL),0
E43E FB                 EI
E43F C9                 RET
                ;
E440 3A3DE5     DSLBNK:LD     A,(SVOFF)
E443 D370               OUT   (PTWOFF),A      }退避していたテキスト·ウィンドウ·オフセットをセットする
E445 C9         PARM0:  RET
                ;
```

```
E446 DB70      SELBNK:IN    A,(PTWOFF)      テキスト・ウィンドウ・オフセットを退避し，ウィンドウが
E448 323DE5           LD    (SVOFF),A       8000H～83FFHを示すようにセット
E44B 3E80             LD    A,IMAGE.R8
E44D D370             OUT   (PTWOFF),A
E44F C9               RET
               ;
E450 EB        JUMP:  EX    DE,HL
E451 E9        JUMP1: JP    (HL)
               ;
               ; graphic library jump table
               ;
E452 45E46F80  JMPTBL:DEFW PARM0,CLS           ;  0
E456 9AE47A80         DEFW PARM1,CLSS          ;  1
E45A 45E47480         DEFW PARM0,CLSH          ;  2
E45E 9AE48F80         DEFW PARM1,ROLUP         ;  3
E462 9AE4A480         DEFW PARM1,ROLUPH        ;  4
E466 9AE4F480         DEFW PARM1,ROLDW         ;  5
E46A 9AE40A81         DEFW PARM1,ROLDWH        ;  6
E46E BDE47E81         DEFW PARM2,ROLLI         ;  7
E472 BDE48581         DEFW PARM2,ROLLIH        ;  8   ジャンプ・テーブル
E476 BDE4CC81         DEFW PARM2,ROLRI         ;  9
E47A BDE4D381         DEFW PARM2,ROLRIH        ; 10
E47E B7E43EE5         DEFW PARM3,PROT          ; 11
E482 B7E47CE5         DEFW PARM3,PROTX         ; 12
E486 B7E49BE5         DEFW PARM3,PROTY         ; 13
E48A 9FE43A82         DEFW PARM4,GCREV         ; 14
E48E 50822DE5         DEFW GCMOVE,PARM5        ; 15
E492 45E4C4E4         DEFW PARM0,KPUTL         ; 16
E496 45E4CDE4         DEFW PARM0,KPUTLH        ; 17
               ;
0012           LIBMAX:EQU  12H                 ;($-JMPTBL)/4
               ;
               ;
E49A           PARM1:                          ; for ROLUP,ROLUPH,ROLDW,
               ;      CLSS                       ROLDWH
E49A 2AF8E5           LD    HL,(ARG0)       ARG0の値をACCにセットする
E49D 7E               LD    A,(HL)
E49E C9               RET
               ;
               ;
E5DA 00        KBANK: DEFB 0
E5DB 0000      KLOC:  DEFW 0
E5DD 0A80      KVECT: DEFW KPUT
               ;
E5DF 0000      PRP:   DEFW 0
               ;
E5E1 0000      VARPTR:DEFW 0
E5E3 00        USRVAL:DEFB 0
               ;
               ;
E5E4                  DEFS 0E5F8H-$
E5F8           PARAM: EQU  $
E5F8           ARG0:  DEFS 2
E5FA           ARG1:  DEFS 2
E5FC           ARG2:  DEFS 2
E5FE           ARG3:  DEFS 2
               ;
               ;
E600                  END
```

〈コメント1〉

漢字1文字PUT

KPUT…………640×200ドットモード時のエントリ・ポイント
KPUTH………640×400ドットモード時のエントリ・ポイント

パラメータ

DEレジスタ……JIS漢字コード（全角文字のみ）
ACC……………漢字PUTを行なうグラフィック・バンクを0～2の値で指定します．ただしこのパラメータはKPUTの場合に有効なパラメータで，KPUTHの場合には無視します．
Bレジスタ………漢字PUTの水平位置を指定します．
Cレジスタ………漢字PUTの垂直位置を指定します．

注意

漢字は1文字16×16ドットのパターンとし，1画面には1行40文字で10行（640×400ドットモードでは20行）表示することとして漢字をPUT位置を定めてあります．そのため行間は4ビット分の間隔があり，また左右の文字間の間隔はありません．

〈コメント2〉

画面消去

CLSH…………640×400ドットモード時の画面消去のエントリ・ポイント
CLS……………640×200ドットモード時の画面消去のエントリ・ポイント

〈コメント3〉

画面のROLL UP，ROLL DOWN

ROLUPH……640×400ドット・モード時のROLL UPのエントリ・ポイント
ROLDWN……640×400ドット・モード時のROLL DOWNのエントリ・ポイント
ROLUP………640×200ドット・モード時のROLL UPのエントリ・ポイント
ROLDW………640×200ドット・モード時のROLL DOWNのエントリ・ポイント

パラメータ

ACC……………ROLL UP及びROLL DOWNを行う行数を与えます．この値は1～199の範囲でなければなりません．

〈コメント4〉

画面のROLL LEFT，ROLL RIGHT

ROLLEH……640×400ドット・モード時のROLL LEFTのエントリ・ポイント
ROLRIH ……640×400ドット・モード時のROLL RIGHTのエントリ・ポイント
ROLLE………640×200ドット・モード時のROLL LEFTのエントリ・ポイント
ROLRI ………640×200ドット・モード時のROLL RIGHTのエントリ・ポイント

パラメータ

ACC……………ROLL LEFTおよびROLL RIGHTを行うドット数を与えます．左右のROLLは8ドット単位で行うため，この値は8～632の範囲でなければなりません．またこの値が8の倍数でない場合には，切り上げを行い8の倍数にします．

〈コメント5〉

●クロス・カーソルの設定と移動
●GCREV……クロス・カーソルの位置の設定と反転表示
パラメータ
DEレジスタ……クロス・カーソルの水平位置
BCレジスタ……クロス・カーソルの垂直位置
IXレジスタ……クロス・カーソルの表示範囲（水平位置）
IYレジスタ……クロス・カーソルの表示範囲（垂直位置）
●GCMOVE……テンキーをスキャンし，押したキーに対応してクロス・カーソルを移動
する戻りパラメータ
DEレジスタ……クロス・カーソルの水平位置
BCレジスタ……クロス・カーソルの垂直位置
ACCレジスタ…クロス・カーソルが移動したか否かのフラグ
移動した場合には0FFH，移動していない場合には0.

〈コメント6〉

パターン反転
PROT………対称軸反転
PROTX……x方向反転
PROTY……y方向反転
パラメータ
DEレジスタ……変換を行うオリジナルのパターンのある配列の最初の要素を指示します．この配列にはBASICのPUT@，GET@で扱う形式でパターンのデータが格納されていなければなりません．
BCレジスタ……変換を行なったパターンを格納する配列の最初の要素を指し示します．この配列の大きさはオリジナル・パターンのある配列と等しいか，それ以上でなければなりません．
注意
(i) このパターン反転機能で扱うパターンは正方形のパターン，つまり縦横のドット数が同じでなければならないません．またこのパターンはBASICのGET@により配列に格納されることとしていますので，代入文等により配列のデータをセットする場合には，格納するパターンのデータ形式に注意して下さい．
(ii) BCレジスタ・ペアが指し示す配列がDEレジスタ・ペアが指し示す配列より小さい場合には，他の変数，システムのプログラムやデータを破壊します．

A.5 インストラクション表の見方

付録にCPUのインストラクション表を載せました．この表は，皆さんのマシン語プログラミングの際の手助けになるように編集したつもりです．

各命令について2通りの表が用意してあります．一つの表は命令セットであり，もう1つの表は，ニーモニック↔マシン語対照表です．ここでは，この2種類の表の見方について説明しましょう．

表A.1および表A.2は8ビットロード命令についてのインストラクション表です．このうち命令セットの方は，その命令の動作，フラグへの影響などが書かれており，それぞれについては以下のようになっています．

■ 命令セットの見方

▶ ニーモニック

ニーモニックの欄は，ザイログ形式のニーモニックが一般的な書式で表現されています．最初の命令を例にとると，実際にはｒおよびｒ′には各8ビットレジスタが指定されて，LD A, C LD B,Aなどとなるのですが，これらをまとめてLD r,rと表現してあります．

▶ シンボリック・オペレーション

シンボリック・オペレーションの欄は，命令の動作を簡単に記号と言葉で説明してあります．LD r,(HL)を例にとると，ｒ←(HL)と書いてあり，HLレジスタ・ペアの示すメモリの内容をｒレジスタにロードすることを意味しています．

▶ フラグ

フラグの欄は，その命令を実行した後の各フラグの変化を示しており，ここに使用した記号は以下に示すような意味があります．

・：変化しない
0：リセット(0になる)される
1：セット(1になる)される
×：不定(1か0かわからない)
↕：演算結果に従った変化をする
P：P／Vフラグが，パリティ・フラグとして動作することを表す
　(1) 偶数パリティ(1の数が偶数)ならセットされる
　(2) 奇数パリティ(1の数が奇数)ならリセットされる
V：P／Vフラグがオーバフロー・フラグとして動作することを表す
　(1) オーバフローがあった場合，つまり8ビット演算において−128(80H)または+127(7FH)を超えたとき，16ビット演算においては−32768(8000H)または+32767(7FFFH)を超えたときセットされる

(2)　オーバフローがなければリセットされる

▶ BINコード／HEXコード

BINコードとHEXコードの欄は，各ニーモニックをBINコードには2進数で，HEXコードには16進数で表現したマシン語が示してあります．LD r,r′を例にとると2進数では10rr′とあり，rおよびr′はそれぞれ3ビットずつ専有しています．ここで備考の欄をみると，r,r′とレジスタの対応表があり，例えばAレジスタなら111，Eレジスタなら011となっています．そこでLD A,Eならrにレジスタ，r′にEレジスタが指定されたことになりますので，rの部分に111，r′の部分に011を当てはめて，2進数で表現されたマシン語は，01111011となります．16進数でこの命令を表現すると7Bとなりますが，HEXコードの欄を見るとLD r,r′の場合40＋となっています．この＋とは，rおよびr′に000が指定された場合のHEXコードを想定していますので，01000000で40となるからです．この他にも，dd, ss, qq, ccなどのオペランドによって変化するビットに関しては，それぞれのビットに0を想定した場合となっていて，＋がついています．

▶ バイト数

バイト数の欄は，各命令が何バイト長であるかを示してあります．例えば，LD r,(IX＋d)は4バイト命令であることがわかります．

▶ Mサイクル

Mサイクル数についての説明は，普段全く気にする必要がないので省略します．

▶ Tステート

Tステート数の欄には，その命令の実行に必要なクロック数が示してあり，1クロックあたりの周期がわかればその命令の実行時間がわかります．例えば，LD r,r′の命令の実行に必要なクロック数は4クロックです．1クロックの周期は，CPUに与えられるクロックが4MHzのとき次のようになります．

$$4\ [\mathrm{MH_z}]=4\times10^{6}\ [\mathrm{H_z}]$$

$$\mathrm{T}=\frac{1}{4\times10^{6}}=2.5\times10^{-7}\ [\mathrm{s}]=250\ [\mathrm{ns}]$$

これより，LD r,r′の実行に必要な時間は，

$$250\ [\mathrm{ns}]\times4=1000\ [\mathrm{ns}]=1\ [\mu\mathrm{s}]$$

当然のことながら，Tステート数が少ない程命令の実行時間は短くなります．また，PC-8801においては，DMAといってCPUの停止している期間や，割込みがありますので，必ずしも計算通りの実行時間にはなりません．

ニーモニック	シンボリック オペレーション	フラグ S Z H P/V N CY	BINコード 76 543 210	HEXコード (基本)	バイト数	Mサイクル数	Tステート数
LD r,r'	r←r'	・・×・×・・・	01 r r'	40+	1	1	4
LD r,n	r←n	・・×・×・・・	00 r 110 ← n →	06+	2	2	7
LD r,(HL)	r←(HL)	・・×・×・・・	01 r 110	46+	1	2	7
LD r,(IX+d)	r←(IX+d)	・・×・×・・・	11 011 101 01 r 110 ← d →	DD 46+	3	5	19
LD r,(IY+d)	r←(IY+d)	・・×・×・・・	11 111 101 01 r 110 ← d →	FD 46	3	5	19
LD (HL),r	(HL)←r	・・×・×・・・	01 110 r	70+	1	2	7
LD (IX+d),r	(IX+d)←r	・・×・×・・・	11 011 101 01 110 r ← d →	DD 70+	3	5	19
LD (IY+d),r	(IY+d)←r	・・×・×・・・	11 111 101 01 110 r ← d →	FD 70+	3	5	19
LD (HL),n	(HL)←n	・・×・×・・・	00 110 110 ← n →	36	2	3	10
LD (IX+d),n	(IX+d)←n	・・×・×・・・	11 011 101 00 110 110 ← d → ← n →	DD 36	4	5	19
LD (IY+d),n	(IY+d)←n	・・×・×・・・	11 111 101 00 110 110 ← d → ← n →	FD 36	4	5	19
LD A,(BC)	A←(BC)	・・×・×・・・	00 001 010	0A	1	2	7
LD A,(DE)	A←(DE)	・・×・×・・・	00 011 010	1A	1	2	7
LD (nn),A	(nn)←A	・・×・×・・・	00 111 010 ← n → ← n →	3A	3	4	13
LD (BC),A	(BC)←A	・・×・×・・・	00 000 010	02	1	2	7
LD (DE),A	(DE)←A	・・×・×・・・	00 010 010	12	1	2	7
LD A,(nn)	A←(nn)	・・×・×・・・	00 110 010 ← n → ← n →	32	3	4	13
LD A,I	A←I	↕ ↕ × 0 × IFF 0 ・	11 101 101 01 010 111	ED 57	2	2	9
LD A,R	A←R	↕ ↕ × 0 × IFF 0 ・	11 101 101 01 011 111	ED 5F	2	2	9
LD I,A	I←A	・・×・×・・・	11 101 101 01 000 111	ED 47	2	2	9
LD R,A	R←A	・・×・×・・・	11 101 101 01 001 111	ED 4F	2	2	9

備考

r,r'	レジスタ
000	B
001	C
010	D
011	E
100	H
101	L
111	A

表A.1　命令セット（LD命令）

■ ニーモニック↔マシン語対照表の見方

ニーモニック↔マシン語対照表は，ハンドアセンブルの際のマシン語早見表となるものです．いくつかの命令を例にして説明しましょう．

×	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	(nn)	n	I	R
LD A,×	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n	ED 57	ED 5F
LD B,×	47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n		
LD C,×	4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n		
LD D,×	57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n		
LD E,×	5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n		
LD H,×	67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n		
LD L,×	6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n		
LD (HL),×	77	70	71	72	73	74	75							36 n		
LD (BC),×	02															
LD (DE),×	12															
LD (IX+d),×	DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n		
LD (IY+d),×	FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n		
LD (nn),×	32 n n															
LD I,×	ED 47															
LD R,×	ED 4F															

表A．2　ニーモニック↔マシン語対照表(LD命令)

▶ LD D,E

まず縦の項からLD D,×を捜します．次に横の項からEを捜して，両方の交差するマスをみると53とあります．つまりこの命令のマシン語は53(もちろん16進数)となります．

▶ LD B,63H

上記と同様にして捜すと，06,nとあり2バイト命令であることがわかります．この場合nに63Hを指定したのでマシン語は0663となります．

▶ LD A,(1234H)

この場合は3A,n,nとなり3バイト命令であることがわかります。nnには1234H番地を指定したのでマシン語は，3A3412となります．

▶ LD (IX－5),10H

DD36dnで4バイト命令であることがわかりますが，ディスプレイスメントには－5を指定したので2の補数で考えてFBとなり，nには10を指定したのでマシン語はDD36FB10となります．

▶ LD B,(DE)

縦と横の交差点を捜してください．空白となっており，この様な命令は存在しないことがわかります．

索　引

A

A……65
ACC……50
ADC……103
ADD……103
AND……36
ASCII……43
ASC(BIN) TO BIN……149
ASC(DEC) TO BCD……154
ASC(DEC) TO BIN……152
ASC(HEX) TO BIN……151

B

B……65
b……85
BASICワーク・エリア……76
BCD……32,33,159,162
BIN TO ASC(BIN)……156
BIN TO ASC(DEC)……158
BIN TO ASC(HEX)……157
BIT……114

C

CALL……107
CALL文……78,137
cc……85
CCF……110
CLEAR文……78
CP……120
CPU……13,41,47
CP／M……207
CRT出力……146
CYフラグ……52

D

D……86
d……66,67
DAA……162
Daisy……206
DB……88
dd……84
DD-8……204
DD-8′……204
DEFB……88
DEFS……89
DEF USR……133
DEFW……88
DI……115
DJNZ……99
DS……89
DUAD-88D……203
DW……88

E

E …… 67,68

e …… 86,99,129

EI …… 115

END …… 87

EQU …… 87

EX …… 96

F

F …… 68

FAC …… 134

Fレジスタ …… 51

G

G …… 68

H

HEXフォーマット …… 205

Hフラグ …… 55

I

I …… 69

IN …… 114

INC …… 98

I／O …… 42

I／Oポート …… 42,112,118,163

J

JR …… 99

L

L …… 69

LD …… 91

LSB …… 21

M

M …… 70

m …… 86

MF-ASM …… 209

MSB …… 21

M系列 …… 191

N

n …… 85

nn …… 86

NOT …… 37

Nフラグ …… 55

N_{88}-BASIC ROMエリア …… 76

O

O …… 71

OR …… 35,115

ORG …… 88

OUT …… 114

P

P …… 71

PC …… 50

POP …… 124

pp …… 85

PUSH …… 124

P／Vフラグ …… 52

Q

qq …… 85

R

R …… 71

r …… 84

RAM …… 41

RET …… 107

ROM …… 41

rr …… 85

RRCA …… 124

RST …… 96

r' …… 84

S

S …… 71

s ……86
SBC …… 105
SCF …… 110
SP ……50
ss……84
SUB …… 105
Sフラグ ……52

T
TM ……71

U
USR関数 …… 133

V
V ……71

W
W……72

X
X ……72
XOR ……38,119,177

Z
Zフラグ ……52

μ
μPD780…… 3,4,7,48,58

ア
アウトプット……71
アキュムレータ……50
アセンブラ……56,204
アセンブリ言語……56
アセンブル……65,217
アドレス……42
アルゴリズム……59
イグザミン・レジスタ……72
インタプリタ……13,15,16
インデックス・レジスタ……50
インプット……69
エディット…… 67,68
演算命令……58
オブジェクト・プログラム……56
オペ・コード……58
オペランド……58

カ
開発ソフト…… 203
加算……27
キースキャン…… 163
記憶装置……41
逆アセンブラ……56,204
グラフィック…… 177
減算……27
ゴー……68
コンパイラ……13,15,16

サ
サブルーチン……60
シフト演算……31
主レジスタ……49
乗算……28
除算……30
シンボル……84
スクリーン・エディタ…… 204
スタック…… 107
スタック・ポインタ……50
制御命令……58
絶対アドレス指定……99
セット・メモリ……71
相対アドレス指定……99
ソース・プログラム……56,217

タ
ダンプ…… 66,67
ディスアセンブル…… 56,69,204

データ・ストローブ信号……168
テキストVRAM……76
テキスト・ウィンドウ……76
テスト・メモリ……71
デバッガ……204
デバッグ……127,132
転送命令……58

ナ
内部ルーチン……140,141
内部ルーチンの解析……187
ニーモニック……55,83
入／出力装置……42
入／出力命令……58

ハ
排他的論理和……38
ハンドアセンブル……127
バイト……21,45,149
番地……42
汎用レジスタ……50
ビジィ信号……168
ビット……21
ビット命令……58
否定……37
ファイル・バッファ……76
フィル・メモリ……68
副レジスタ……49
プリンタ出力……148,167
プリントスイッチ……71
フローチャート……59
プログラム・カウンタ……50
ブロック命令……58
分岐命令……58
ベース……65
ベリファイ・テープ……71
変数エリア……76
補数……27

マ
マシン語領域……78
ムーブ・メモリ……70
命令長……57
メモリ……41,193
メモリマップ……75
文字列サーチ……174
モニタ……63
モニタ・コマンド……64

ラ
ライト・テープ……72
ライン入力……145
乱数の発生……191
リアルタイムのキー判定……144
リード・テープ……71
レジスタ……42,48
レジスタ構成……48
ローダ……204
論理演算……34,37,39
論理積……36
論理和……35

ワ
割込み……197

1文字入力……142

16ビット加減算命令……110

2進数の用語……21

(X)……86

参考文献

〔１〕「PC-8801ユーザーズ・マニュアル」
NEC発行

〔２〕「PC-8801 BASICリファレンスマニュアル」
NEC発行

〔３〕「マイクロコンピュータのプログラミング」
石田晴久編　共立出版発行

〔４〕「PC-8801グラフィックスのすべて」
工藤丈彦，榊正憲，田村明史，横溝和宏共著
アスキー発行

〔５〕「PC-Techknow 8801 Vol.1」
栗山浩一，平松達雄，松尾篤弥共著
アスキー発行

協力　榊正憲・工藤丈彦

PC-8801 マシン語入門

アスキー・システム・バンク(PC88＃3)

1983年6月20日　第1版第1刷発行
定価2,500円

著　者　藤井敬雄・民田雅人
発行者　塚本慶一郎
発行所　株式会社アスキー
　〒107　港区南青山5－11－5住友南青山ビル5F
振　替　東京4-161144
電　話　03－486－7111(代表)

©1983　ASCII Corporation. Printed in Japan.

本書は著作権法上の保護を受けています．本書の一部あるいは全部について（ソフトウェア及びプログラムを含む），株式会社アスキーから文書による許諾を得ずに，いかなる方法においても無断で複写，複製することは禁じられています．

編集担当　土屋信明
制作協力　株式会社ワトム
印刷　壮光舎印刷株式会社

ISBN4-87148-702-4　C3055　¥2500E

付録
インストラクション表

アスキー・システム・バンク(PC88＃3)
PC-8801 マシン語入門
藤井敬雄・民田雅人共著

アスキー出版局

8080Aとμ PD780　CPUの命令対応表

8080A ニーモニック	μPD780 ニーモニック	8080A ニーモニック	μPD780 ニーモニック	8080A ニーモニック	μPD780 ニーモニック
ACI	ADC A,n	IN	IN A,(n)	POP H	POP HL
ADC M	ADC A,(HL)	INR M	INC (HL)	POP PSW	POP AF
ADC r	ADC A,r	INR r	INC r	PUSH B	PUSH BC
ADD M	ADD A,(HL)	INX B	INC BC	PUSH D	PUSH DE
ADD r	ADD A,r	INX D	INC DE	PUSH H	PUSH HL
ADI	ADD A,n	INX H	INC HL	PUSH PSW	PUSH AF
ANA M	AND (HL)	INX SP	INC SP	RAL	RLA
ANA r	AND r	JC	JP C,nn	RAR	RRA
ANI	AND n	JM	JP M,nn	RC	RET C
CALL	CALL nn	JMP	JP nn	RET	RET
CC	CALL C,nn	JNC	JP NC,nn	RLC	RLCA
CM	CALL M,nn	JNZ	JP NZ,nn	RM	RET M
CMA	CPL	JP	JP P,nn	RNC	RET NC
CMC	CCF	JPE	JP PE,nn	RNZ	RET NZ
CMP M	CP(HL)	JPO	JP PO,nn	RP	RET P
CMP r	CP r	JZ	JP Z,nn	RPE	RET PE
CNC	CALL NC,nn	LDA	LD A,(nn)	RPO	RET PO
CNZ	CALL NZ,nn	LDAX B	LD A,(BC)	RRC	RRCA
CP	CALL P,nn	LDAX D	LD A,(DE)	RST	RST p
CP E	CALL PE,nn	LHLD	LD HL,(nn)	RZ	RET Z
CPI	CP n	LXI B	LD BC,nn	SBB M	SBC A,(HL)
CPO	CALL PO,nn	LXI D	LD DE,nn	SBB r	SBC A,r
CZ	CALL Z,nn	LXI H	LD HL,nn	SBI	SBC A,n
DAA	DAA	LXI SP	LD SP,nn	SHLD	LD (nn),HL
DAD B	ADD HL,BC	MVI M	LD (HL),n	SPHL	LD SP,HL
DAD D	ADD HL,DE	MVI r	LD r,n	STA	LD (nn),A
DAD H	ADD HL,HL	MOV M,r	LD (HL),r	STAX B	LD (BC),A
DAD SP	ADD HL,SP	MOV r,M	LD r,(HL)	STAX D	LD (DE),A
DCR M	DEC (HL)	MOV r1,r2	LD r,r′	STC	SCF
DCR r	DEC r	NOP	NOP	SUB M	SUB (HL)
DCX B	DEC BC	ORA M	OR (HL)	SUB r	SUB r
DCX D	DEC DE	ORA r	OR r	SUI	SUB n
DCX H	DEC HL	ORI	OR n	XCHG	EX DE,HL
DCX SP	DEC SP	OUT	OUT (n),A	XRA M	XOR (HL)
DI	DI	PCHL	JP (HL)	XRA r	XOR r
EI	EI	POP B	POP BC	XRI	XOR n
HLT	HALT	POP D	POP DE	XTHL	EX (SP),HL

レジスター覧

レジスタ	レジスタ名	ビット数	レジスタ	レジスタ名	ビット数
A	アキュムレータ	8	F	フラグレジスタ	8
B	Bレジスタ	8	C	Cレジスタ	8
D	Dレジスタ	8	E	Eレジスタ	8
H	Hレジスタ	8	L	Lレジスタ	8
A′	アキュムレータ	8	F′	フラグレジスタ	8
B′	B′レジスタ	8	C′	C′レジスタ	8
D′	D′レジスタ	8	E′	E′レジスタ	8
H′	H′レジスタ	8	L′	L′レジスタ	8
I	割込みベクトルレジスタ	8	R	メモリリフレッシュレジスタ	7
IX	インデックスレジスタIX	16	IY	インデックスレジスタIY	16
SP	スタック・ポインタ	16	PC	プログラム・カウンタ	16

フラグへの影響

インストラクション	$D_7D_6D_5D_4D_3D_2D_1D_0$ S Z　H　P/V N CY	コメント
ADD A,s ; ADC A,s	↕ ↕ × ↕ × V 0 ↕	8ビット加算命令
SUB s ; SBC A,s ; CP s ; NEG	↕ ↕ × ↕ × V 1 ↕	8ビット減算・比較・NEG命令
AND s	↕ ↕ × 1 × P 0 0	論理演算命令
OR s ; XOR s	↕ ↕ × 0 × P 0 0	
INC s	↕ ↕ × ↕ × V 0 ・	8ビット・インクリメント命令
DEC s	↕ ↕ × ↕ × V 1 ・	8ビット・デクリメント命令
ADD dd,ss	・ ・ × × × ・ 0 ↕	16ビット加算命令
ADC HL,ss	↕ ↕ × × × V 0 ↕	16ビット加算命令(キャリー含む)
SBC HL,ss	↕ ↕ × × × V 1 ↕	16ビット減算命令(キャリー含む)
RLA ; RLCA ; RRA ; RRCA	・ ・ × 0 × ・ 0 ↕	ローテイト・アキュムレータ命令
RLS ; RLC s ; RR s ; RRC s	↕ ↕ × 0 × P 0 ↕	ローテイト・シフト命令
SLA s ; SRA s ; SRL s		
RLD ; RRD	↕ ↕ × 0 × P 0 ・	ローテイト・デジット命令
DAA	↕ ↕ × ↕ × P ・ ↕	10進補正演算命令
CPL	・ ・ × 1 × ・ 1 ・	アキュムレータ反転命令
SCF	・ ・ × 0 × ・ 0 1	キャリーセット命令
CCF	・ ・ × × × ・ 0 ↕	キャリー反転命令
IN r,(C)	↕ ↕ × ↕ × P 0 ・	入力ポート指定命令
INI ; IND ; OUTI ; OUTD	× ↕ × × × × 1 ・	ブロック I/O命令 B ≠ 0ならばZ=0，その他はZ=1
INIR ; INDR ; OTIR ; OTDR	× 1 × × × × 1 ・	
LDI ; LDD	× × × 0 × ↕ 0 ・	ブロック転送命令 BC ≠ 0ならばP/V=1，その他はP/V=0
LDIR ; LDDR	× × × 0 × 0 0 ・	
CPI ; CPIR	× ↕ × × × ↕ 1 ・	ブロック・サーチ命令 A=(HL)ならばZ=1，その他はZ=0 BC ≠ 0ならばP/V=1，その他はP/V=0
CPD ; CPDR		
LD A,I ; LD A,R	↕ ↕ × 0 × IFF 0 ・	IFFの内容をP/Vにコピー
BIT b,s	× ↕ × 1 × × 0 ・	Sのビットbの内容をZにコピー

シンボル	意味	シンボル	意味
S	サイン・フラグ	×	不定
Z	ゼロ・フラグ	r	CPUレジスタ(A, B, C, D, E, H, L)
P/V	パリティ/オーバフロー・フラグ	s	8ビット・ロケーション
H	ハーフ・キャリー・フラグ	ss	16ビット・ロケーション
N	加/減算・フラグ	ii	インデックスレジスタ
CY	キャリー/リンク・フラグ	R	リフレッシュ・カウンタ
↕	フラグが演算結果の影響を受ける	n	8ビットの数値
・	フラグが演算結果の影響を受けない	nn	16ビットの数値
0	フラグが演算結果の影響によってリセットされる	IFF	IFFの値がP/Vフラグの値となる
1	フラグが演算結果の影響によってセットされる		

8ビット・ロード

×	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	(nn)	n	I	R
LD A,×	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n	ED 57	ED 5F
LD B,×	47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n		
LD C,×	4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n		
LD D,×	57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n		
LD E,×	5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n		
LD H,×	67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n		
LD L,×	6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n		
LD (HL),×	77	70	71	72	73	74	75							36 n		
LD (BC),×	02															
LD (DE),×	12															
LD (IX+d),×	DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n		
LD (IY+d),×	FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n		
LD (nn),×	32 n n															
LD I,×	ED 47															
LD R,×	ED 4F															

ニーモニック	シンボリックオペレーション	フラグ S Z　H　P/V N CY	BINコード 76　543　210	HEXコード (基本)	バイト数	Mサイクル数	Tステート数	備考
LD r,r′	r←r′	・ ・ × ・ × ・ ・ ・	01　r　r′	40+	1	1	4	r, r′　レジスタ 000　B 001　C 010　D 011　E 100　H 101　L 111　A
LD r,n	r←n	・ ・ × ・ × ・ ・ ・	00　r　110 ←　n　→	06+	2	2	7	
LD r,(HL)	r←(HL)	・ ・ × ・ × ・ ・ ・	01　r　110	46+	1	2	7	
LD r,(IX+d)	r←(IX+d)	・ ・ × ・ × ・ ・ ・	11　011　101 01　r　110 ←　d　→	DD 46+	3	5	19	
LD r,(IY+d)	r←(IY+d)	・ ・ × ・ × ・ ・ ・	11　111　101 01　r　110 ←　d　→	FD 46	3	5	19	
LD (HL),r	(HL)←r	・ ・ × ・ × ・ ・ ・	01　110　r	70+	1	2	7	
LD (IX+d),r	(IX+d)←r	・ ・ × ・ × ・ ・ ・	11　011　101 01　110　r ←　d　→	DD 70+	3	5	19	
LD (IY+d),r	(IY+d)←r	・ ・ × ・ × ・ ・ ・	11　111　101 01　110　r ←　d　→	FD 70+	3	5	19	
LD (HL),n	(HL)←n	・ ・ × ・ × ・ ・ ・	00　110　110 ←　n　→	36	2	3	10	
LD (IX+d),n	(IX+d)←n	・ ・ × ・ × ・ ・ ・	11　011　101 00　110　110 ←　d　→ ←　n　→	DD 36	4	5	19	
LD (IY+d),n	(IY+d)←n	・ ・ × ・ × ・ ・ ・	11　111　101 00　110　110 ←　d　→ ←　n　→	FD 36	4	5	19	
LD A,(BC)	A←(BC)	・ ・ × ・ × ・ ・ ・	00　001　010	0A	1	2	7	
LD A,(DE)	A←(DE)	・ ・ × ・ × ・ ・ ・	00　011　010	1A	1	2	7	
LD A,(nn)	A←(nn)	・ ・ × ・ × ・ ・ ・	00　111　010 ←　n　→ ←　n　→	3A	3	4	13	
LD (BC),A	(BC)←A	・ ・ × ・ × ・ ・ ・	00　000　010	02	1	2	7	
LD (DE),A	(DE)←A	・ ・ × ・ × ・ ・ ・	00　010　010	12	1	2	7	
LD (nn),A	(nn)←A	・ ・ × ・ × ・ ・ ・	00　110　010 ←　n　→ ←　n　→	32	3	4	13	
LD A,I	A←I	↕ ↕ × 0 × IFF 0 ・	11　101　101 01　010　111	ED 57	2	2	9	
LD A,R	A←R	↕ ↕ × 0 × IFF 0 ・	11　101　101 01　011　111	ED 5F	2	2	9	
LD I,A	I←A	・ ・ × ・ × ・ ・ ・	11　101　101 01　000　111	ED 47	2	2	9	
LD R,A	R←A	・ ・ × ・ × ・ ・ ・	11　101　101 01　001　111	ED 4F	2	2	9	

16ビット・ロード

×	AF	BC	DE	HL	SP	IX	IY	nn	(nn)
LD AF, ×									
LD BC, ×								01 n n	ED 4B n n
LD DE, ×								11 n n	ED 5B n n
LD HL, ×								21 n n	2A n n
LD SP, ×				F9		DD F9	FD F9	31 n n	ED 7B n n
LD IX, ×								DD 21 n n	DD 2A n n
LD IY, ×								FD 21 n n	FD 2A n n
LD (nn), ×		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n		
PUSH ×	F5	C5	D5	E5		DD E5	FD E5		
POP ×	F1	C1	D1	E1		DD E1	FD E1		

ニーモニック	シンボリック オペレーション	フラグ S Z H P/V N CY	BINコード 76 543 210	HEXコード (基本)	バイト数	Mサイクル数	Tステート数	備考
LD dd, nn	dd←nn	· · × · × · · ·	00 dd0 001 ← n → ← n →	01+	3	3	10	dd レジスタ 00 BC 01 DE 10 HL 11 SP
LD IX, nn	IX←nn	· · × · × · · ·	11 011 101 00 100 001 ← n → ← n →	DD 21	4	4	14	
LD IY, nn	IY←nn	· · × · × · · ·	11 111 101 00 100 001 ← n → ← n →	FD 21	4	4	14	
LD HL, (nn)	H←(nn+1) L←(nn)	· · × · × · · ·	00 101 010 ← n → ← n →	2A	3	5	16	nnは2バイト数下位1バイトはBINコードの直後上位1バイトはその次に入る。
LD dd, (nn)	dd_H←(nn+1) dd_L←(nn)	· · × · × · · ·	11 101 101 01 dd1 011 ← n → ← n →	ED 4B+	4	6	20	
LD IX, (nn)	IX_H←(nn+1) IX_L←(nn)	· · × · × · · ·	11 011 101 00 101 010 ← n → ← n →	DD 2A	4	6	20	
LD IY, (nn)	IY_H←(nn+1) IY_L←(nn)	· · × · × · · ·	11 111 101 00 101 010 ← n → ← n →	FD 2A	4	6	20	
LD (nn), HL	(nn+1)←H (nn)←L	· · × · × · · ·	00 100 010 ← n → ← n →	22	3	5	16	
LD (nn), dd	(nn+1)←dd_H (nn)←dd_L	· · × · × · · ·	11 101 101 01 dd0 011 ← n → ← n →	ED 43+	4	6	20	
LD (nn), IX	(nn+1)←IX_H (nn)←IX_L	· · × · × · · ·	11 011 101 00 100 010 ← n → ← n →	DD 22	4	6	20	
LD (nn), IY	(nn+1)←IY_H (nn)←IY_L	· · × · × · · ·	11 111 101 00 100 010 ← n → ← n →	FD 22	4	6	20	
LD SP, HL	SP←HL	· · × · × · · ·	11 111 001	F9	1	1	6	
LD SP, IX	SP←IX	· · × · × · · ·	11 011 101 11 111 001	DD F9	2	2	10	
LD SP, IY	SP←IY	· · × · × · · ·	11 111 101 11 111 00	FD F9	2	2	10	qq レジスタ 00 BC 01 DE 10 HL 11 AF
PUSH qq	(SP−2)←qq_L (SP−1)←qq_H	· · × · × · · ·	11 qq0 101	C5+	1	3	11	
PUSH IX	(SP−2)←IX_L (SP−1)←IX_H	· · × · × · · ·	11 011 101 11 100 101	DD E5	2	4	15	
PUSH IY	(SP−2)←IY_L (SP−1)←IY_H	· · × · × · · ·	11 111 101 11 100 101	FD E5	2	4	15	
POP qq	qq_H←(SP+1) qq_L←(SP)	· · × · × · · ·	11 qq0 001	C1+	1	3	10	
POP IX	IX_H←(SP+1) IX_L←(SP)	· · × · × · · ·	11 011 101 11 100 001	DD E1	2	4	14	
POP IY	IY_H←(SP+1) IY_L←(SP)	· · × · × · · ·	11 111 101 11 100 001	FD E1	2	4	14	

交換・ブロック転送・ブロックサーチ

交換

EX AF, AF′	08
EX DE, HL	EB
EX (SP), HL	E3
EX (SP), IX	DD E3
EX (SP), IY	FD E3
EXX	D9

ブロック転送

LDI	ED A0
LDIR	ED B0
LDD	ED A8
LDDR	ED B8

ブロックサーチ

CPI	ED A1
CPIR	ED B1
CPD	ED A9
CPDR	ED B9

ニーモニック	シンボリック オペレーション	フラグ S Z H P/V N CY	BINコード 76 543 210	HEXコード (基本)	バイト数	Mサイクル数	Tステート数	備考
EX DE, HL	DE↔HL	· · × · × · · ·	11 101 011	EB	1	1	4	
EX AF, AF′	AF↔AF′	· · × · × · · ·	00 001 000	08	1	1	4	
EXX	$\begin{pmatrix}BC\\DE\\HL\end{pmatrix} \leftrightarrow \begin{pmatrix}BC'\\DE'\\HL'\end{pmatrix}$	· · × · × · · ·	11 011 001	D9	1	1	4	レジスタの切り換え
EX (SP), HL	H↔(SP+1) L↔(SP)	· · × · × · · ·	11 100 011	E3	1	5	19	
EX (SP), IX	IX_H↔(SP+1) IX_L↔(SP)	· · × · × · · ·	11 011 101 11 100 011	DD E3	2	6	23	
EX (SP), IY	IY_H↔(SP+1) IY_L↔(SP)	· · × · × · · ·	11 111 101 11 100 011	FD E3	2	6	23	
LDI	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1	· · × 0 × ↕ 0 · ①	11 101 101 10 100 000	ED A0	2	4	16	ポインタ 1増 バイト・カウンタ 1減
LDIR	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1	· · × 0 × 0 0 ·	11 101 101 10 110 000	ED B0	2	5	21	BC≠0のとき
	BC＝0ならば終わり				2	4	16	BC＝0のとき
LDD	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1	· · × 0 × ↕ 0 · ①	11 101 101 10 101 000	ED A8	2	4	16	
LDDR	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1	· · × 0 × 0 0 ·	11 101 101 10 111 000	ED B8	2	5	21	BC≠0のとき
	BC＝0ならば終わり				2	4	16	BC＝0のとき
CPI	A−(HL) HL←HL+1 BC←BC−1	↕ ↕ × ↕ × ↕ 1 · ② ①	11 101 101 10 100 001	ED A1	2	4	16	
CPIR	A−(HL) HL←HL−1 BC←BC−1	↕ ↕ × ↕ × ↕ 1 · ② ①	11 101 101 10 110 001	ED B1	2	5	21	BC≠0かつ A≠(HL)のとき
	A＝(HL)または BC＝0ならば終わり				2	4	16	BC＝0か A＝(HL)のとき
CPD	A−(HL) HL←HL+1 BC←BC−1	↕ ↕ × ↕ × ↕ 1 · ② ①	11 101 101 10 101 001	ED A9	2	4	16	
CPDR	A−(HL) HL←HL−1 BC←BC−1	↕ ↕ × ↕ × ↕ 1 · ② ①	11 101 101 10 111 001	ED B9	2	5	21	BC≠0かつ A≠(HL)のとき
	A＝(HL)または BC＝0ならば終わり				2	4	16	BC＝0か A＝(HL)のとき

① BC−1＝0ならばP/Vは0, その他は1　② A＝(HL)ならばZ＝1, その他は0

8ビット算術・論理演算

×	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
ADD A, ×	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADC A, ×	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUB ×	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SBC A, ×	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
AND ×	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
XOR ×	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
OR ×	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
CP ×	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INC ×	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DEC ×	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

ニーモニック	シンボリックオペレーション	フラグ	BINコード	HEXコード	バイト数	Mサイクル数	Tステート数	備考
		S Z H P/V N CY	76 543 210	(基本)				
ADD A, r	A←A+r	↕ ↕ × ↕ × V 0 ↕	10 k r	80+	1	1	4	r レジスタ 000 B 001 C 010 D 011 E 100 H 101 L 111 A
ADD A, n	A←A+n	↕ ↕ × ↕ × V 0 ↕	11 k 110 ← n →	C6+	2	2	7	
ADD A, (HL)	A←A+(HL)	↕ ↕ × ↕ × V 0 ↕	10 000 110	86	1	2	7	
ADD A, (IX+d)	A←A+(IX+d)	↕ ↕ × ↕ × V 0 ↕	11 011 101 10 k 110 ← d →	DD 86+	3	5	19	
ADD A, (IY+d)	A←A+(IY+d)	↕ ↕ × ↕ × V 0 ↕	11 111 101 10 k 110 ← d →	FD 86+	3	5	19	ニーモニック k ADD 000 ADC 001 SUB 010 SBC 011 AND 100 OR 110 XOR 101 CP 111
ADC A, s	A←A+s+CY	↕ ↕ × ↕ × V 0 ↕	上記ADDを基本形としそれぞれ4種類ある(備考参照)		※1 1 2 1 3	※1 1 2 2 5	※1 4 7 7 19	
SUB s	A←A−s	↕ ↕ × ↕ × V 1 ↕						
SBC A, s	A←A−s−CY	↕ ↕ × ↕ × V 1 ↕						
AND s	A←A∧s	↕ ↕ × 1 × P 0 0						
OR s	A←A∨s	↕ ↕ × 0 × P 0 0						
XOR s	A←A⊕s	↕ ↕ × 0 × P 0 0						
CP s	A−s	↕ ↕ × ↕ × V 1 ↕						
INC r	r←r+1	↕ ↕ × ↕ × V 0 ·	00 r ℓ	00+	1	1	4	S=r, n, (HL), (IX+d), (IY+d)
INC (HL)	(HL)←(HL)+1	↕ ↕ × ↕ × V 0 ·	00 110 ℓ	30+	1	3	11	
INC (IX+d)	(IX+d)←(IX+d)+1	↕ ↕ × ↕ × V 0 ·	11 011 101 00 110 ℓ ← d →	DD 30+	3	6	23	
INC (IY+d)	(IY+d)←(IY+d)+1	↕ ↕ × ↕ × V 0 ·	11 111 101 00 110 ℓ ← d →	FD 30+	3	6	23	ニーモニック ℓ INC 100 DEC 101
DEC m	m←m−1	↕ ↕ × ↕ × V 1 ·	上記INCを基本形とし4種類ある		※2 1 1 3 3	※2 1 3 6 6	※2 4 11 23 23	m=r, (HL), (IX+d), (IY+d)

※1 sによって決まる
※2 mによって決まる

16ビット算術演算

×	BC	DE	HL	SP	IX	IY
ADD HL, ×	09	19	29	39		
ADD IX, ×	DD 09	DD 19		DD 39	DD 29	
ADD HL, ×	FD 09	FD 19		FD 39		FD 29
ADC HL, ×	ED 4A	ED 5A	ED 6A	ED 7A		
SBC HL, ×	ED 42	ED 52	ED 62	ED 72		
INC ×	03	13	23	33	DD 23	FD 23
DEC ×	0B	1B	2B	3B	DD 2B	FD 2B

ニーモニック	シンボリックオペレーション	フラグ	BINコード	HEXコード	バイト数	Mサイクル数	Tステート数	備考
		S Z H P/V N CY	76 543 210	(基本)				
ADD HL, ss	HL←HL+ss	· · × × × · 0 ↕	00 ss1 001	09+	1	3	11	ss レジスタ 00 BC 01 DE 10 HL 11 SP
ADC HL, ss	HL←HL+ss+CY	↕ ↕ × × × V 0 ↕	11 101 101 01 ss1 010	ED 4A+	2	4	15	
SBC HL, ss	HL←HL−ss−CY	↕ ↕ × × × V 1 ↕	11 101 101 01 ss0 010	ED 42+	2	4	15	
ADD IX, pp	IX←IX+pp	· · × × × · 0 ↕	11 011 101 00 pp1 001	DD 09+	2	4	15	pp レジスタ 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY←IY+rr	· · × × × · 0 ↕	11 111 101 00 rr1 001	FD 09+	2	4	15	
INC ss	ss←ss+1	· · × · × · · ·	00 ss0 011	03+	1	1	6	
INC IX	IX←IX+1	· · × · × · · ·	11 011 101 00 100 011	DD 23	2	2	10	
INC IY	IY←IY+1	· · × · × · · ·	11 111 101 00 100 011	FD 23	2	2	10	rr レジスタ 00 BC 01 DE 10 IY 11 SP
DEC ss	ss←ss−1	· · × · × · · ·	00 ss1 011	0B+	1	1	6	
DEC IX	IX←IX−1	· · × · × · · ·	11 011 101 00 101 011	DD 2B	2	2	10	
DEC IY	IY←IY−1	· · × · × · · ·	11 111 101 00 101 011	FD 2B	2	2	10	

ジャンプ

×	無条件	キャリー	ノンキャリー	ゼロ	ノンゼロ	パリティ偶数	パリティ奇数	負	正	カウント
		CY	NC	Z	NZ	PE	PO	M	P	
JP ×,nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JR ×,e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
JP (HL)	E9									
JP (IX)	DD E9									
JP (IY)	FD E9									
DJNZ e										10 e-2

ニーモニック	シンボリックオペレーション	フラグ	BINコード	HEXコード	バイト数	Mサイクル数	Tステート数	備考
		S Z H P/V N CY	76 543 210	(基本)				
JP nn	PC←nn	・・×・×・・・	11 000 011 ← n → ← n →	C3	3	3	10	cc 条件 000 NZ 001 Z 010 NC 011 CY 100 PO 101 PE 110 P 111 M NZ:ノンゼロ Z:ゼロ CY:キャリー PO:パリティ奇数 PE:パリティ偶数 P:正 M:負
JP cc,nn	ccが真ならば PC←nn, その他は次へ	・・×・×・・・	11 cc 010 ← n → ← n →	C2+	3	3	10	
JP e	PC←PC+e	・・×・×・・・	00 011 000 ← e-2 →	18	2	3	12	
JR C,e	C=1ならばPC←PC+e	・・×・×・・・	00 111 000 ← e-2 →	38	2	3	12	
	C=0ならば次へ				2	2	7	
JR NC,e	C=0ならばPC←PC+e	・・×・×・・・	00 110 000 ← e-2 →	30	2	3	12	
	C=1ならば次へ				2	2	7	
JR Z,e	Z=1ならばPC←PC+e	・・×・×・・・	00 101 000 ← e-2 →	28	2	3	12	
	Z=0ならば次へ				2	2	7	
JR NZ,e	Z=0ならばPC←PC+e	・・×・×・・・	00 100 000 ← e-2 →	20	2	3	12	
	Z=1ならば次へ				2	2	7	
JP (HL)	PC←HL	・・×・×・・・	11 101 001	E9	1	1	4	
JP (IX)	PC←IX	・・×・×・・・	11 011 101 11 101 001	DD E9	2	2	8	
JP (IY)	PC←IY	・・×・×・・・	11 111 101 11 101 001	FD E9	2	2	8	
DJNZ e	B←B−1 B≠0ならばPC←PC+1	・・×・×・・・	00 010 000 ← e-2 →	10	2	3	13	
	B=0ならば次へ				2	2	8	

コール・リターン

コール・リターン命令

×	無条件	キャリー	ノンキャリー	ゼロ	ノンゼロ	パリティ偶数	パリティ奇数	負	正
		CY	NC	Z	NZ	PE	PO	M	P
CALL ×,nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n
RET ×	C9	D8	D0	C8	C0	E8	E0	F8	F0
PETI	ED 4D								
RETN	ED 45								

リスタート

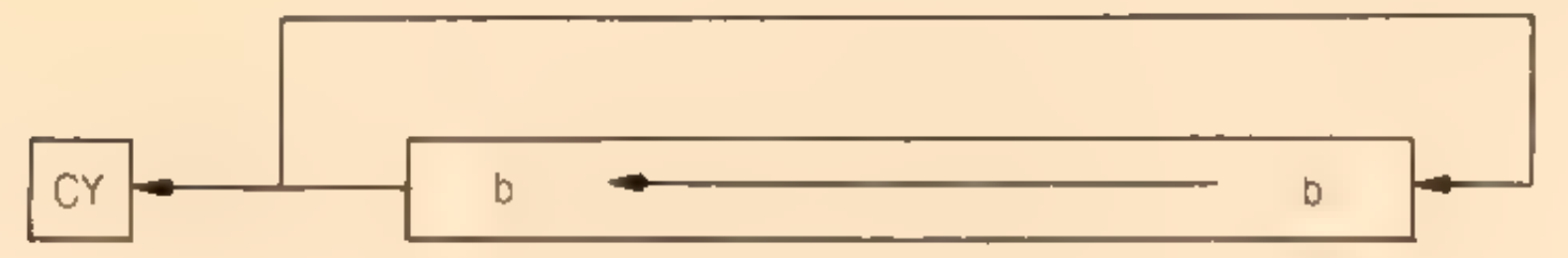
×	0000H	0008H	0010H	0018H	0020H	0028H	0030H	0038H
RST ×	C7	CF	D7	DF	E7	EF	F7	FF

ニーモニック	シンボリックオペレーション	フラグ	BINコード	HEXコード	バイト数	Mサイクル数	Tステート数	備考
		S Z H P/V N CY	76 543 210	(基本)				
CALL nn	(SP−1)←PC_H (SP−2)←PC_L PC←nn	・・×・×・・・	11 001 101 ← n → ← n →	CD	3	5	17	cc 条件 000 NZ 001 Z 010 NC 011 CY 100 PO 101 PE 110 P 111 M NZ:ノンゼロ Z:ゼロ CY:キャリー PO:パリティ奇数 PE:パリティ偶数 P:正 M:負 t p 000 00H 001 08H 010 10H 011 18H 100 20H 101 28H 110 30H 111 38H
CALL cc,nn	ccが真ならば CALL nnと同じ	・・×・×・・・	11 cc 100 ← n → ← n →	C4+	3	5	17	
	その他ならば次へ				3	3	10	
RET	PC_L←(SP) PC_H←(SP+1)	・・×・×・・・	11 001 001	C9	1	3	10	
RET cc	ccが真ならば RETと同じ	・・×・×・・・	11 cc 000	C0+	1	3	11	
	その他ならば次へ				1	1	5	
RET I	割込みからのリターン	・・×・×・・・	11 101 101 01 001 101	ED 4D	2	4	14	
RET N	ノン・マスカラブル 割込みからのリターン	・・×・×・・・	11 101 101 01 000 101	ED 45	2	4	14	
RST p	(SP−1)←PC_H (SP−2)←PC_L PC_H←0 PC_L←p	・・×・×・・・	11 t 111	C7+	1	3	11	

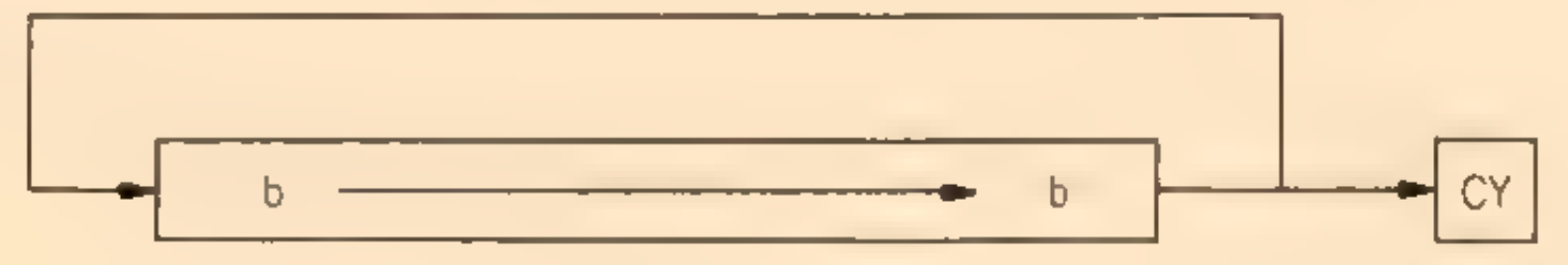
ローテイトとシフト I

×	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
RLC ×	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
RRC ×	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
RL ×	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
RR ×	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 26	FD CB d 1E
SLA ×	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 1E	FD CB d 26
SRA ×	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD 1E d 2E	FD CB d 2E
SRL ×	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
RLD								ED 6F		
RRD								ED 67		

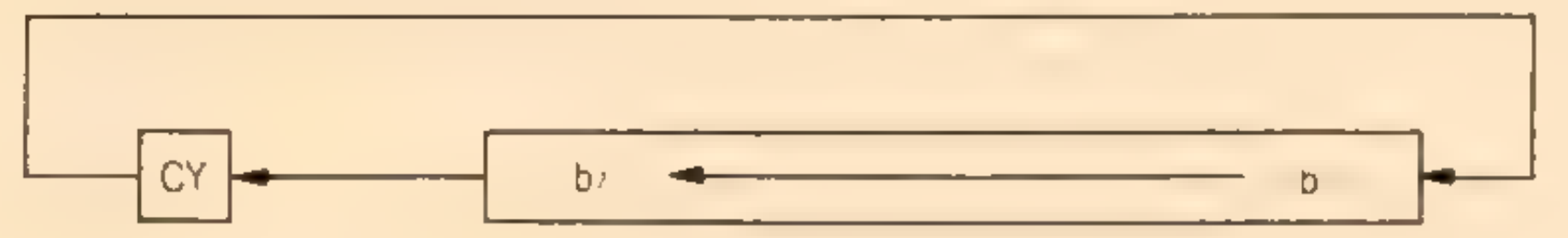
RLCA	07
RRCA	0F
RLA	17
RRA	1F



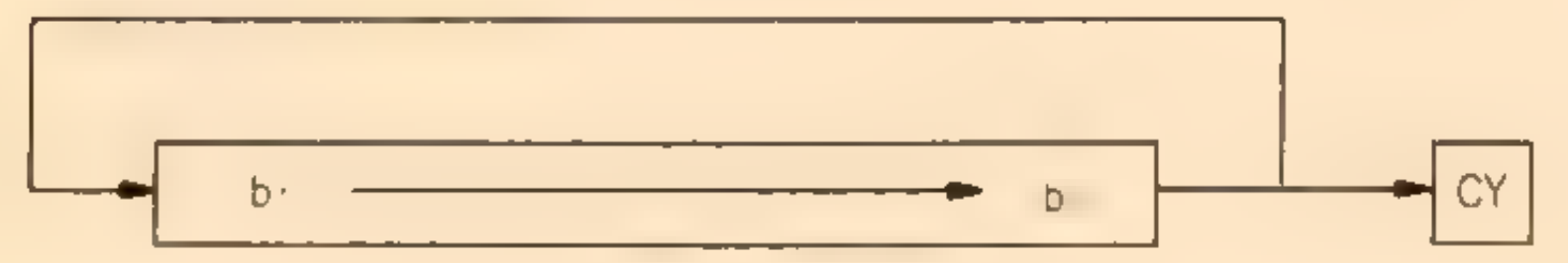
左ローテイト・サーキュラ



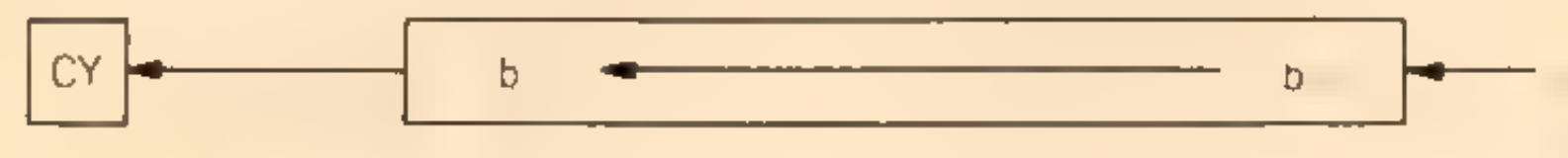
右ローテイト・サーキュラ



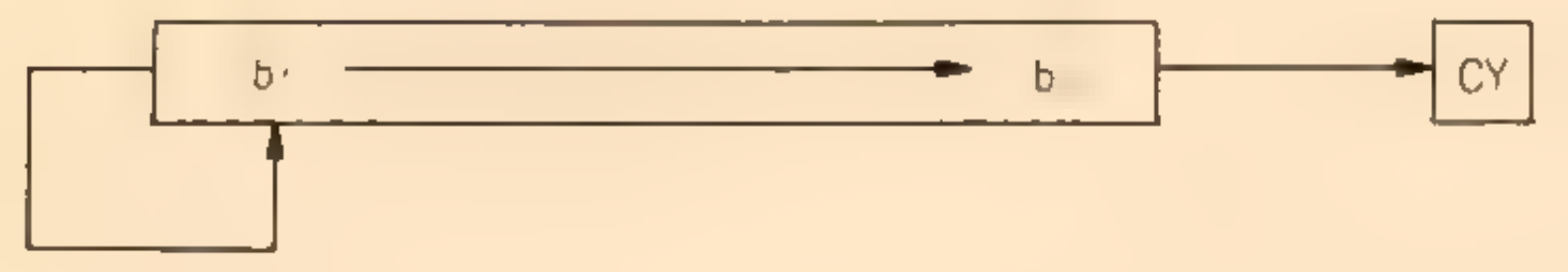
左ローテイト



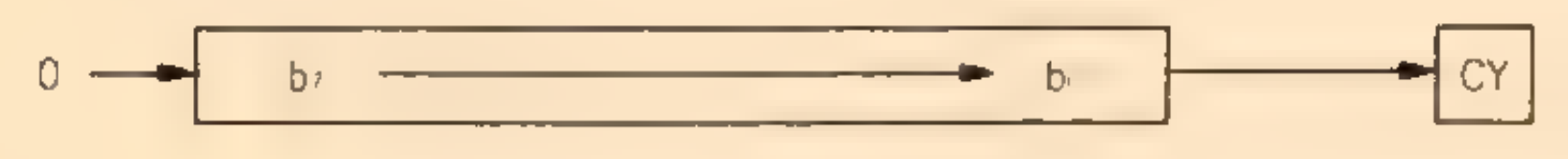
右ローテイト



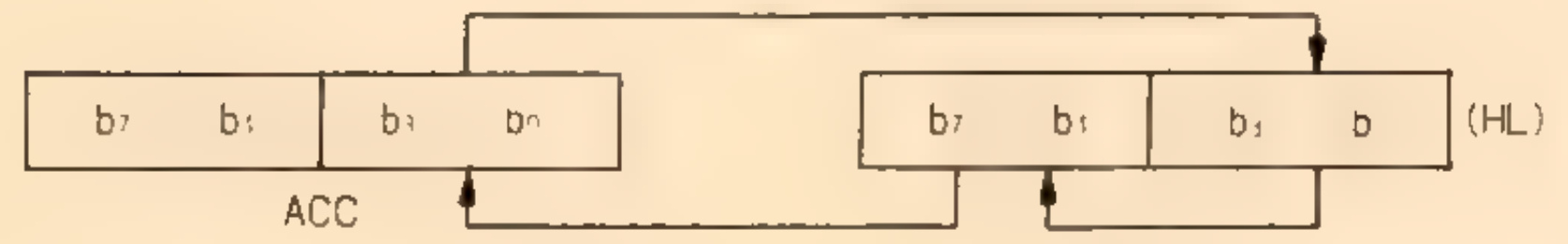
算術左シフト



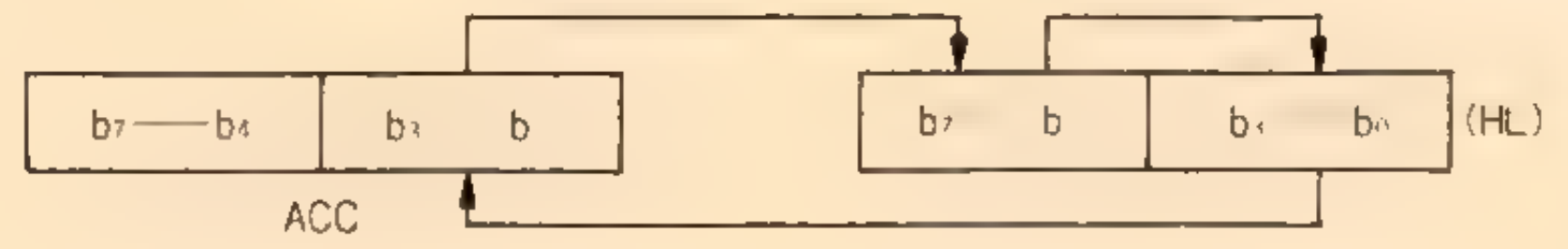
算術右シフト



論理右シフト

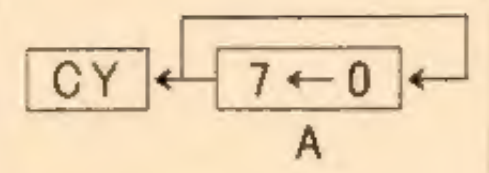
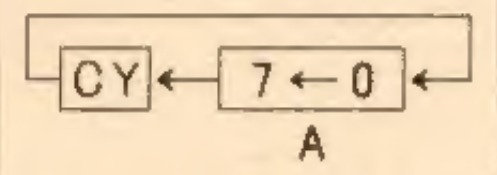
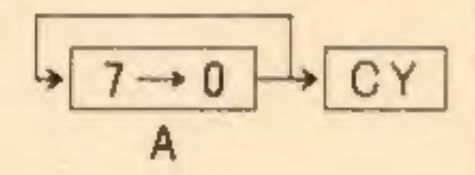
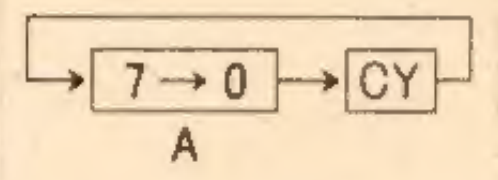
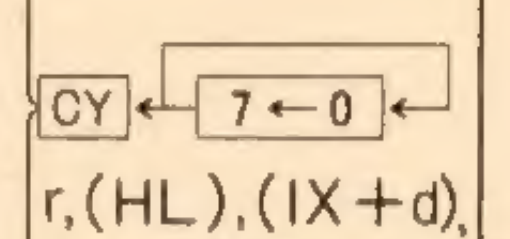
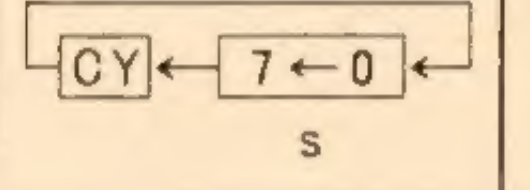
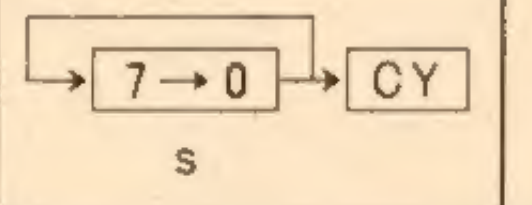
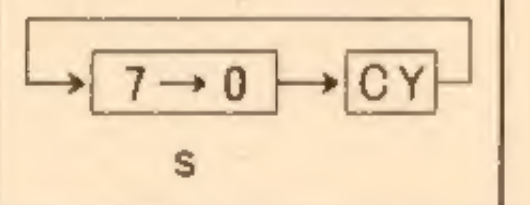
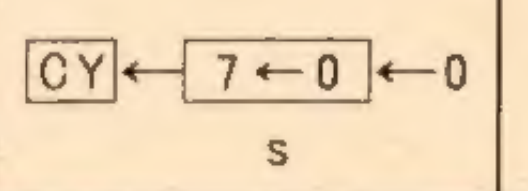
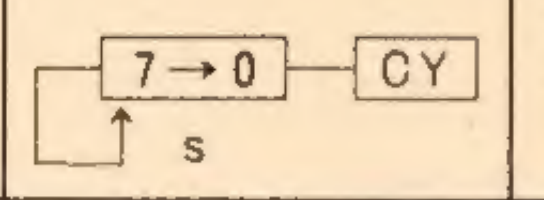
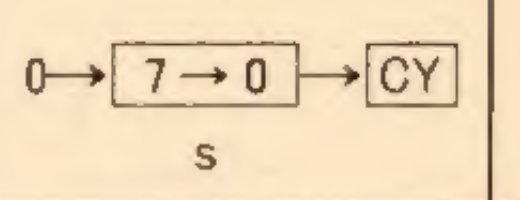
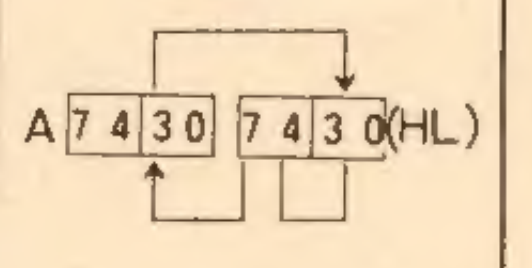
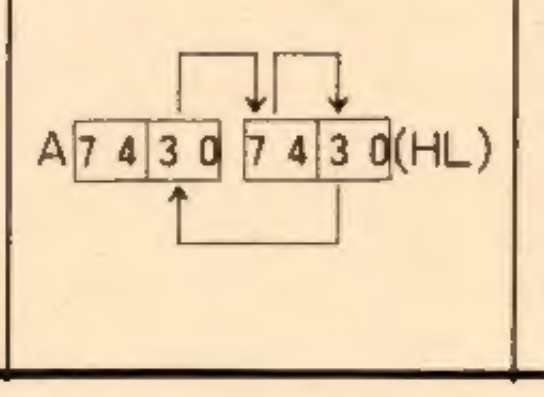


左ローテイト・デジット



右ローテイト・デジット

ローテイトとシフト II

ニーモニック	シンボリックオペレーション	フラグ	BINコード	HEXコード	バイト数	Mサイクル数	Tステート数	備考
		S Z H P/V N CY	76 543 210	(基本)				
RLCA		· · × 0 × · 0 ↕	00 000 111	07	1	1	4	左ローテイト・サーキュラ・アキュムレータ
RLA		· · × 0 × · 0 ↕	00 010 111	17	1	1	4	左ローテイト・アキュムレータ
RRCA		· · × 0 × · 0 ↕	00 001 111	0F	1	1	4	右ローテイト・サーキュラ・アキュムレータ
RRA		· · × 0 × · 0 ↕	00 011 111	1F	1	1	4	右ローテイト・アキュムレータ
RLC r	 r, (HL), (IX+d), (IY+d)	↕ ↕ × 0 × P 0 ↕	11 001 011 00 k r	CB 00+	2	2	8	左ローテイト・サーキュラ・レジスタ r
RLC (HL)		↕ ↕ × 0 × P 0 ↕	11 001 011 00 k 110	CB 06+	2	4	15	
RLC (IX+d)		↕ ↕ × 0 × P 0 ↕	11 011 101 11 001 011 ← d → 00 k 110	DD CB 06+	4	6	23	
RLC (IY+d)		↕ ↕ × 0 × P 0 ↕	11 111 101 11 001 011 ← d → 00 k 110	FD CB 06+	4	6	23	
RL s		↕ ↕ × 0 × P 0 ↕	上記RLCを基本形とし それぞれ 4種類ある (備考参照)		※ 2 2 4 4	※ 2 4 6 6	※ 8 15 23 23	S=r, (HL), (IX+d), (IY+d)
RRC s		↕ ↕ × 0 × P 0 ↕						
RR s		↕ ↕ × 0 × P 0 ↕						
SLA s		↕ ↕ × 0 × P 0 ↕						
SRA s		↕ ↕ × 0 × P 0 ↕						
SRL s		↕ ↕ × 0 × P 0 ↕						
RLD		↕ ↕ × 0 × P 0 ·	11 101 101 01 101 111	ED 6F	2	5	18	左ローテイト・デジット・アキュムレータ, (HL) アキュムレータ 上位 4ビット変化せず
RRD		↕ ↕ × 0 × P 0 ·	11 101 101 01 100 111	ED 67	2	5	18	右ローテイト・デジット・アキュムレータ, (HL) アキュムレータ 上位 4ビット変化せず

r	レジスタ
0 0 0	B
0 0 1	C
0 1 0	D
0 1 1	E
1 0 0	H
1 0 1	L
1 1 1	A

ニーモニック	k
RLC	0 0 0
RRC	0 0 1
RL	0 1 0
RR	0 1 1
SLA	1 0 0
SRA	1 0 1
SRL	1 1 1

※s によって決まる

ビット操作

×	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
BIT 0, ×	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
BIT 1, ×	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
BIT 2, ×	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
BIT 3, ×	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
BIT 4, ×	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
BIT 5, ×	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
BIT 6, ×	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
BIT 7, ×	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RES 0, ×	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
RES 1, ×	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
RES 2, ×	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
RES 3, ×	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
RES 4, ×	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
RES 5, ×	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
RES 6, ×	CB B7	CB 30	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
RES 7, ×	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET 0, ×	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
SET 1, ×	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
SET 2, ×	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
SET 3, ×	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
SET 4, ×	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
SET 5, ×	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
SET 6, ×	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
SET 7, ×	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

ニーモニック	シンボリックオペレーション	フラグ	BINコード	HEXコード	バイト数	Mサイクル数	Tステート数
		S Z H P/V N CY	76 543 210	(基本)			
BIT b, r	$Z \leftarrow \overline{r_b}$	× ↕ × 1 × × 0 ·	11 001 011 01 b r	CB 40+	2	2	8
BIT b, (HL)	$Z \leftarrow \overline{(HL)}_b$	× ↕ × 1 × × 0 ·	11 001 011 01 b 110	CB 46+	2	3	12
BIT b, (IX+d)	$Z \leftarrow \overline{(IX+d)}_b$	× ↕ × 1 × × 0 ·	11 011 101 11 001 011 ← d → 01 b 110	DD CB 46+	4	5	20
BIT b, (IY+d)	$Z \leftarrow \overline{(IY+d)}_b$	× ↕ × 1 × × 0 ·	11 111 101 11 001 011 ← d → 01 b 110	FD CB 46+	4	5	20
SET b, r	$r_b \leftarrow 1$	· · × · × · · ·	11 001 011 a b r	CB	2	2	8
SET b, (HL)	$(HL)_b \leftarrow 1$	· · × · × · · ·	11 001 011 a b 110	CB 06+	2	4	15
SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	· · × · × · · ·	11 011 101 11 001 011 ← d → a b 110	DD CB 06+	4	6	23
SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	· · × · × · · ·	11 111 101 11 001 011 ← d → a b 110	FD CB 06+	4	6	23
RES b, m	$S_b \leftarrow 0$ m≡r, (HL), (IX+d), (IY+d)	· · × · × · · ·	上記 SETを基本形として 4種ある		※ 2 2 4 4	※ 2 4 6 6	※ 8 15 23 23

備考

r	レジスタ
0 0 0	B
0 0 1	C
0 1 0	D
0 1 1	E
1 0 0	H
1 0 1	L
1 1 1	A

b	テスト・ビット
0 0 0	0
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7

ニーモニック	a
SET	1 1
RES	1 0

※ mによって決まる

入出力

入力

×	A	B	C	D	E	H	L
IN ×, (n)	D B n						
IN ×, (C)	E D 7 8	E D 4 0	E D 4 8	E D 5 0	E D 5 8	E D 6 0	E D 6 8

INI	E D A 2	ブロック入力コマンド
INIR	E D B 2	
IND	E D A A	
INDR	E D B A	

出力

×	A	B	C	D	E	H	L
OUT (n), ×	D 3 n						
OUT (c), ×	E D 7 9	E D 4 1	E D 4 9	E D 5 1	E D 5 9	E D 6 1	E D 6 9

OUT I	E D A 3	ブロック出力コマンド
OT I R	E D B 3	
OUTD	E D A B	
OTDR	E D B B	

ニーモニック	シンボリック オペレーション	フラグ S Z H P/V N CY	BINコード 76 543 210	HEXコード (基本)	バイト数	Mサイクル数	Tステート数	備考
IN A, (n)	A←(n)	・・×・×・・・	11 011 011 ← n →	D B	2	3	11	n→A_0～A_7 Acc→A_8～A_{15}
IN r, (C)	r←(C)	↕↕×↕×P0・ r=110の時のみフラグ変化	11 101 101 01 r 000	E D 40+	2	3	12	
INI	(HL)←(C) B←B−1 HL←HL+1	×↕××××1・ ①	11 101 101 10 100 010	E D A 2	2	4	16	C→A_0～A_7 B→A_8～A_{15}
INIR	(HL)←(C) B←B−1 HL←HL+1 B=0まで繰り返す	×1××××1・	11 101 101 10 110 010	E D B 2	2 2	5 B≠0のとき 4 B=0のとき	21 16	r レジスタ 0 0 0 B 0 0 1 C 0 1 0 D 0 1 1 E 1 0 0 H 1 0 1 L 1 1 1 A
IND	(HL)←(C) B←B−1 HL←HL−1	×↕××××1・ ①	11 101 101 10 101 010	E D A A	2	4	16	
INDR	(HL)←(C) B←B−1 HL←HL−1 B=0まで繰り返す	×1××××1・	11 101 101 10 111 010	E D B A	2 2	5 B≠0のとき 4 B=0のとき	21 16	
OUT (n), A	(n)←A	・・×・×・・・	11 010 011 ← n →	D 3	2	3	11	n→$(A-BUS)_{0\sim7}$ Acc→$(A\text{-}BUS)_{8\sim15}$
OUT (C), r	(C)←r	・・×・×・・・	11 101 101 01 r 001	E D 41+	2	3	12	
OUTI	(C)←(HL) B←B−1 HL←HL+1	×↕××××1・ ①	11 101 101 10 100 011	E D A 3	2	4	16	C→A_0～A_7 B→A_8～A_{15}
OTIR	(C)←(HL) B←B−1 HL←HL+1 B=0まで繰り返す	×1××××1・	11 101 101 10 110 011	E D B 3	2 2	5 B≠0のとき 4 B=0のとき	21 16	
OUTD	(C)←(HL) B←B−1 HL←HL−1	×↕××××1・ ①	11 101 101 10 101 011	E D A B	2	4	16	
OTDR	(C)←(HL) B←B−1 HL←HL−1 B=0まで繰り返す	×1××××1・	11 101 101 10 111 011	E D B B	2 2	5 B≠0のとき 4 B=0のとき	21 16	

①B−1=0ならばZ=1, その他Z=0

汎用算術演算・CPU制御命令

汎用算術演算

DAA	2 7
CPL	2 F
NEG	E D 4 4
CCF	3 F
SCF	3 7

CPU制御命令

NOP	0 0	
HALT	7 6	
DI	F 3	
EI	F B	
IM 0	E D 4 6	8080Aモード
IM 1	E D 5 6	38H番地へのコール命令
IM 2	E D 5 E	レジスタIと割込みデバイスからの 8ビットデータを用いた時, 間接コール命令

ニーモニック	シンボリック オペレーション	フラグ S Z H P/V N CY	BINコード 76 543 210	HEXコード (基本)	バイト数	Mサイクル数	Tステート数	備考
DAA	10進補正 (加算・減算)	↕↕×↕×P・↕	00 100 111	2 7	1	1	4	デシマル・アジャスト・アキュムレータ
CPL	A←$\overline{A}$	・・×1×・1・	00 101 111	2 F	1	1	4	1の補数Acc
NEG	A←0−A	↕↕×↕×V1↕	11 101 101 01 000 100	E D 4 4	2	2	8	2の補数Acc
CCF	CY←$\overline{CY}$	・・×××・0↕	00 111 111	3 F	1	1	4	キャリーの反転
SCF	CY←1	・・×0×・01	00 110 111	3 7	1	1	4	キャリー・セット
NOP	No operation	・・×・×・・・	00 000 000	0 0	1	1	4	
HALT	CPU待機	・・×・×・・・	01 110 110	7 6	1	1	4	
DI	IFF←0	・・×・×・・・	11 110 011	F 3	1	1	4	ディセーブル割込み
EI	IFF←1	・・×・×・・・	11 111 011	F B	1	1	4	イネーブル割込み
IM 0	MODE 0にセット	・・×・×・・・	11 101 101 01 000 110	E D 4 6	2	2	8	割込みモードのセット
IM 1	MODE 1にセット	・・×・×・・・	11 101 101 01 010 110	E D 5 6	2	2	8	
IM 2	MODE 2にセット	・・×・×・・・	11 101 101 01 011 110	E D 5 E	2	2	8	

ASCII SYSTEM BANK

PC88第③巻 内容抜粋

- マシン語とは何か
- ２進・16進演算
- モニタ機能をさぐる
- メモリマップとマシン語領域
- ニーモニックの表現形式
- 擬似命令とソース・リスト
- メモリの内容交換
- キャラクタ・スクウェア
- グラフィック模様を描く
- USR関数,CALL文の使い方
- 内部ルーチンの利用
- 内部ルーチンの解析と実際
- データの型と変換
- I／Oポートを使う
- 文字列のサーチ
- テキスト・エリアを読む
- グラフィック・パターンを動かす
- シンボル・データの作成
- 乱数をつくる
- スピードとメモリ効率を上げる
- 割込みについて
- マシン語プログラム開発ソフト
- DUAD88Dの機能
- 多言語システムCP／Mについて
- MF-ASM(エムエフ・アセンブラ)
- MF-ASM文法と使用法
- Glib(グラフィックライブラリ)の解析
- インストラクション表の見方
- インストラクション表

藤井敬雄・民田雅人共著

アスキー・システム・バンク(PC88#3)

PC-8801 マシン語入門

定価2,500円
ISBN4-87148-702-4
C3055 ¥2500E